NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MULTIPLE PATH STATIC ROUTING PROTOCOLS
FOR PACKET SWITCHED NETWORKS

by

Harry Thornberry Schiantarelli

Septebmer 1983

Thesis Advisor: J. M. Wozencraft

Approved for public release; distribution unlimited

T216785
T216786

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle): Multiple Path Static Routing Protocols for Packet Switched Networks	5. TYPE OF REPORT & PERIOD COVERED: Master's Thesis: September 1983
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Harry Thornberry Schiantarelli	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Naval Postgraduate School, Monterey, California 93943	12. REPORT DATE: September 1983
	13. NUMBER OF PAGES: 142
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Packet Switched Networks; Routing Protocols; Simulation; Statis Routing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

A central issue in the design of a packet switched communications network is the development of an efficient routing protocol, which determines the routes followed by the information as it traverses the network. The performance of a static routing protocol that allows for multiple path routing based on the use of routing fractions, is analyzed using computer simulation. Comparison is made between the performance of this protocol and

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

S/N 0102-LF-014-6601

that of a minimum number of hops routing protocol. Routing fractions calculated by two different methods are used and their relative performance analyzed. A comparative analysis is done of the performance of this protocol when using virtual circuit service and datagram service in the network. Provision is made to extend use of the simulation program to analyze dynamic routing cases using routing fractions.

Approved for public release; distribution unlimited.

Multiple Path Static Routing Protocols for Packet Switched Networks

by

Harry Thornberry Schiantarelli
Lieutenant Commander, Peruvian Navy
B.S., Naval Telecommunications School, Peru, 1974

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN TELECOMMUNICATIONS SYSTEMS MANAGEMENT

and

MASTER OF SCIENCE IN ENGINEERING SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1983

ABSTRACT

A central issue in the design of a packet switched communications network is the development of an efficient routing protocol, which determines the routes followed by the information as it traverses the network. The performance of a static routing protocol that allows for multiple path routing based on the use of routing fractions, is analyzed using computer simulation. Comparison is made between the performance of this protocol and that of a minimum number of hops routing protocol. Routing fractions calculated by two different methods are used and their relative performance analyzed. A comparative analysis is done of the performance of this protocol when using virtual circuit service and datagram service in the network. Provision is made to extend use of the simulation program to analyze dynamic routing cases using routing fractions.

TABLE OF CONTENTS

I. INTRODUCTION 10
A. GENERAL 10
B. COMMUNICATION NETWORKS 11

II. PACKET SWITCHED NETWORKS 13
A. GENERAL 13
B. DATAGRAM AND VIRTUAL-CIRCUIT SERVICES 16
C. NETWORK LAYERS AND PROTOCOLS 17

III. ROUTING 22
A. DEFINITION 22
B. STATIC AND DYNAMIC ROUTING 24
C. DISTRIBUTED AND CENTRALIZED ROUTING 27
D. FLOW CONTROL AND CONGESTION CONTROL 30
E. ROUTING FRACTIONS 36
F. LOOP AVOIDANCE 39

IV. SIMULATION OF A PACKET SWITCHED COMMUNICATIONS NETWORK 41
A. MODEL CHARACTERISTICS 41
B. PARAMETERS OF THE PACKET SWITCHED NETWORK . . 45
C. COMPUTER LANGUAGE 49
D. SIMULATION PROGRAM DESCRIPTION 50
E. SIMULATION OUTPUTS DESCRIPTION 58
1. INITIAL REPORT 59
2. PERIOD REPORT 61
3. PERIOD DATA 63
4. NETWORK REPORT 64
5. NETWORK DATA 66

V. CONCLUSIONS AND RECOMMENDATIONS 67
A. CONCLUSIONS 67

B. RECOMMENDATIONS FOR FUTURE STUDY 73

APPENDIX A: GENERATION OF GEOMETRICALLY DISTRIBUTED NUMBER OF PACKETS 75

APPENDIX B: LINK NAME ASSIGNMENT 77

APPENDIX C: MINIMUM NUMBER OF HOPS SINGLE PATH ROUTING TABLE 79

APPENDIX D: GRAPHIC REPRESENTATION OF DATA COLLECTED DURING SIMULATION 84

APPENDIX E: SIMULATION PROGRAM 91

APPENDIX F: PROGRAM TO GRAPH GENERAL STATISTICS OF THE SIMULATION RUN 120

APPENDIX G: PROGRAM TO GRAPH PERIODICAL STATISTICS OF THE SIMULATION RUN 129

LIST OF REFERENCES 138

BIBLIOGRAPHY 141

INITIAL DISTRIBUTION LIST 142

LIST OF TABLES

I. GRANULARITY COMPARISON 68

II. PERFORMANCE COMPARISON 70

III. PERFORMANCE COMPARISON 73

IV. ROUTING TABLE 83

LIST OF FIGURES

3.1	THROUGHPUT AND TRANSIT DELAY VERSUS OFFERED LOAD	34
3.2	DELAY VERSUS OFFERED LOAD	35
4.1	NETWORK TOPOLOGY	49
C.1	SINK-TREE FOR EXTERNAL VERTICES	80
C.2	SINK-TREE FOR INTERNAL VERTICES	81
C.3	SINK-TREE FOR CENTRAL NODE	82
D.1	PKTS STILL IN NETWORK FOR ROUTING-FRACTION PROTOCOL	85
D.2	PKTS. STILL IN NETWORK FOR MINIMUM-HOPS PROTOCOL	86
D.3	COMPARISON OF PACKETS STILL IN THE NETWORK	87
D.4	COMPARISON OF MAXIMUM LINK USAGE FOR THE NETWORK	88
D.5	COMPARISON OF AVERAGE QUEUE LENGTH FOR THE NETWORK	89
D.6	COMPARISON OF AVG. DELAY FOR COMPLETING TRIP PKTS	90

ACKNOWLEDGEMENTS

I wish to express my thanks to Professor John M. Wozencraft for his guidance and patience throughout the course of this thesis work. His broad knowledge on the subject of packet switched networks and of communications in general made this effort a valuable experience.

This work is dedicated to my wife Gabriela and my daughter Gaby, whose constant support and the atmosphere of love they created made it all possible.

I. INTRODUCTION

A. GENERAL

In recent years we have been experiencing a technological convergence of computers and telecommunications in such a way that it has become difficult to separate what is processing and what is communication. The constant improvement in computer processing speed and storage capacity together with their reduction in price and size, have definitively influenced this change, making it increasingly attractive to use programmable computers to control communication networks and process the information transmitted through them. This allows the integration of traditional communications with computer networks and distributed systems, while using the same means of transmission, and thus represents a more efficient use of the communication channels. This situation creates the necesity for the development of new transmission techniques that can accomodate the requirements of higher data rates, as well as procedures to handle larger volumes of information.

A kind of network that is becoming increasingly popular is the Packet Switched network. Crucial to the design of this kind of network is the development of efficient routing procedures, which determine the routes followed by the information as it traverses the network. The present study represents further work in that direction and as such involves the analysis of a routing procedure that uses so called Routing Fractions.

B. COMMUNICATION NETWORKS

There are two basic types of communication network architectures: Point to Point and Broadcast. In point to point architectures the network normally contains numerous channels (we will call them links), which may be implemented by a band of frequencies, cable, optical fiber, etc, connecting a pair of stations (we will call them nodes). If two nodes which are not connected by a link wish to communicate, they must do so indirectly via one or more intermediate nodes. This characteristic allows for a wide variety of network topologies. It is also an important feature when interconnecting nodes that are far apart from each other, by allowing the use of a variety of alternate routes in case of link failure or congestion situations.

In a point to point network, if the links selected for the communication are used exclusively by the two nodes for the duration of the communication, the technique is called "line switching". In this case, both the nodes and interconnecting links must be free before the information can start to be sent. A classical example of the use of this technique is the public telephone network where all the links between the caller and recipient of the call will be held for their exclusive use, until the end of the communication. If one or more of these links or the node called is busy, the call setup cannot be completed, and a busy tone is received at the calling node to indicate the procedure must be restarted.

If instead, the links can be shared by two or more communications, and the information that is received at an intermediate node may be stored there until the required output link is free and then forwarded, the technique is called "store and forward". The node caller (source node) does not have to wait for all the links involved in the

communication, or the destination node, to be free before starting to send the information. In most cases this allows for a more efficient use of resources. Other advantages of point to point architectures include their ability to support simultaneous transfer of information between different stations in the network, and the use of high speed links to speed-up this transfer.

In broadcast architectures, there is a single communication link shared by all nodes, so that information sent by any node is received by all other nodes. Satellite and Bus architectures are some examples of this type of networks. Since there is only one path for all communications, the total overall transfer rate of information is limited by the baniwith and speed (capacity) of that single channel, and its failure can cause complete system failure. Another limitation of this architecture is interference between stations requesting the use of the channel, so that some kind of contention-resolving or polling scheme must be used, making it more complex and less efficient.

Broadcast architectures are used mainly in local area networks (LAN). There the communication problem is restricted to short distances and usually higher capacity links (1-10 megabits per second) are available. Examples of this kind of architecture are Ethernet from Xerox Corporation, and Hyperbus from Network System Corporation. In this thesis we will concentrate on store-and-forward networks and in particular on so called "Packet Switched" networks.

II. PACKET SWITCHED NETWORKS

A. GENERAL

Perhaps the most popular kind of store and forward network today is the packet switched network. In these networks the communications, or messages, are subdivided into fixed length parts called "Packets" (usually the packet size is around 1000 bits). Packets are transmitted through the network using the best route selected by the routing protocols (see Chapter III), wich exists at the time the packet is released into the network. Each packet traverses the network intermixed with packets of other messages.

The main advantage of packet switched networks over line switched networks is the conservation (reduction) of switching time. With line switching, a complete path of communication must be set up between the source and destination node before the communication begins. Path setup is established through a signaling process. Before transmission of a message, a reservation signal is sent towards the destination. While traveling node by node to the destination node, the signal reserves links along the path. If at any intermediate node it cannot find a free link, it waits for a link to become free while holding the links it has reserved so far. When a link becomes free it reserves it and goes to the next node to repeat the same process. When the signal reaches the destination node, a path has been reserved between source and destination nodes. As shown by Kermani and Kleinrock in [Ref. 1], as the input traffic increases, a network using line switching saturates very quickly and the message delay increases rapidly. This rapid saturation is the result of the overhead imposed by the

switches to set the path, which are slow relative to the duration of a message. Because of the reservation operation, and the exclusive use of a path, line switching is not a good choice particularly if traffic load is high, since it causes a substantial decrease in the network carrying capacity. Therefore, for communication networks which have many nodes and require the exchange of large number of messages, the principles used in packet switched networks can provide a better communication scheme.

Packet switched networks support apparent full connectivity among the nodes by virtue of the store and forward nature of the system. Typically, packet switched networks use fewer than (3N/2) bi-directional (duplex) links (where N is the number of nodes), whereas a fully connected network requires (N x (N-1)/2) duplex links. Packet switched networks allow traffic to be concentrated on higher speed or lower cost links, as required to reduce channel costs and/or transmission delays. They also allow sending traffic through alternate routes and around failed or congested links or nodes, increasing channel utilization, reducing transmission delay, and making the network more robust and dependable.

Packet switched networks require that each node be able to process certain information about the packet itself, and about the network structure and status, in order to assign the correct routing to the packet. Due to this requirement and the further need for high speed processing to operate efficiently, each node in the network is implemented with a programmable computer that controls its operation and regulates its participation in the network.

Packet switched networks allow for the transmission of both data and digitized voice communications, but some special considerations must be taken when doing this. Delays in data communications are not as critical as in

voice communications, and allow the storage and further reassembly of the packets to recreate the original message (as long as all the packets corresponding to a message eventually arrive to the destination node). Nevertheless, bit errors or packet losses are intolerable for data communications, and a procedure for error detection/correction is required for proper operation. In contrast, voice communications allow for some level of bit error and even packet loss, since the receiver (the human brain) will in most cases interpolate and make up for the errors. Actual experiments show that when the lost packets constitute less than 1% of the offered voice traffic, confortable conversation can be maintained [Ref. 2]. On the other hand voice communications require near real-time processing, and do not accept out of order packets, making it necessary to implement some kind or priority scheme that will give precedence to voice packets over data packets, and a routing procedure that will ensure the delivery of packets in the same order they were generated. Speech packets must meet a strict overall delay constraint of less than 200-500 miliseconds [Ref. 3] for comfortable conversations, this makes it impractical to store the packets of a message in the receiving node until the last one arrives and then to reorder them; it also constrains the maximum number of links they can traverse. This constraint must be taken into account, particularly when designing the network topology and/or assigning link capacities, in order to ensure that packet-time delay does not render voice communications useless.

B. DATAGRAM AND VIRTUAL-CIRCUIT SERVICES

There are two basic kinds of "service" that a packet switched network can provide, in reference to the way it delivers its packets: Datagram service,and Virtual Circuit service. In datagram service the packets, which are referred to as "datagrams", are treated as independent entities, that is, each packet of a message is routed and delivered independently of the other packets of the same message. There is no enforced relationship between the order in which one node enters datagrams into the network and the order in which these same datagrams arrive at their destination node. The use of this scheme requires each datagram to contain the whole information necesary for its routing. Therefore, each packet will typically contain in its header the destination node, the message to which it belongs, packet number, and any other information required by the particular routing scheme implemented.

By contrast, the virtual circuit is a sequenced service, that is, the order in which the packets are entered into the network is the same order in which they arrive at their destination node. This is guaranteed by requiring that all the packets of a message follow the same route as the first packet of that message. In this case each packet needs only to carry in its header an identification of the virtual circuit to which it belongs, and the different nodes will automatically route it thorough the links that correspond to the virtual circuit.

A virtual circuit resembles the line switching technique in the sense that all packets of a message traverses the same path, but differs from it in that more than one virtual circuit may simultaneously include the same link as part of the route for its packets and thus share the use of that link. Another important difference is that if a link or

node fails, the packet may be routed through alternate routes and communication is not lost as it would occur with line switching.

Each virtual circuit requires an explicit setup procedure (generally done by the first packet of a message or a "request-to-send" packet) which establishes the route, followed by a data transfer and an explicit shutdown procedure. Once the "circuit" has been set, individual packets do not have to carry their destination address, since it was specified during setup, allowing them to carry more data. Packetized virtual circuits emerge as a promising approach to pure voice packet networks [Ref. 4].

Virtual circuits appear to the end users as if they were dedicated lines; however, within the network many different virtual circuits share the same communication links. The network model that we chose for this study can be configured to provide either virtual circuit or datagram service. This characteristic allows us to compare the behavior of the routing algorithm for each case.

Some examples of networks that provide these kind of services are IBM's System Network Architecture (SNA), which provides virtual circuit service, and DEC's Digital Network Architecture (DNA), which provides datagram service.

C. NETWORK LAYERS AND PROTOCOLS

In order to simplify design complexity, most networks are organized as a series of layers or levels, each one built upon its predecesor. The purpose of each layer is to offer certain services to the higher layers, shielding them from the details of how the services offered are actually implemented. The rules and conventions that govern the timing and forward of data exchange between layers are called protocols, and the set of layers and protocols are

called the network architecture. The basic aims of a network architecture are that the network has its functions clearly defined and applications and communications are separated. At the same time, the applications interface to the network must be as simple as possible.

There is no unique layered structure, but for this study we will use the one adopted by the International Standards Organization (ISO), which is a seven layer network model called the "Reference Model of Open Systems Interconnection" also called OSI [Ref. 5]. The different layers of this model are:

(1) Physical Layer
(2) Data Link Layer
(3) Network Layer
(4) Transport Layer
(5) Session Layer
(6) Presentation Layer
(7) Application Layer

The Physical Layer deals with the actual transmission of the raw bits over the communication link. It is the set of electrical or mechanical functions required to establish, maintain and release physical connections between the nodes and transmision circuits (links). Its main purpose is to ensure that if a bit 1 was sent, the receiver gets a 1 and not a 0. No consideration is taken with the information content of the bits, nor with character or frame boundaries.

The Data Link Layer moves one step away from the Physical Layer, and its purpose is to present the network with an error free communication link. This layer establishes, maintains and releases a link connection between two nodes. This involves the division of data into frames, and the corresponding mechanisms to transmit them sequentially, detect any error in their transmission, and process the acknowledgment sent back by the receiver. Since

the Physical Layer merely transmits a stream of bits without regard to their meaning or structure, it is up to the data link layer to create and recognize frame boundaries.

The next layer, the Network Layer, represents the main concern of this thesis. It determines the main characteristics of the units of information, or packets, and how they are exchanged and routed through the network. This layer is also called the Communications Subnet layer, since it receives messages from the originator, divides them into packets and ensures that they are correctly received at their destination, and in the proper order. It masks all switching and direction consideration from the higher level layers. It is in this layer that the kind of service the network will provide is determined (virtual circuit or datagram), and thus it contains the corresponding procedures to handle it. Routing protocols are therefore the heart of the network, and their effectiveness and efficiency will affect the overall performance of the network, probably more than any other protocol.

The next four layers, have to do with the way the different processes in each node communicate between them, and with processes in other nodes (Transport and Session Layer); and the interfaces with the actual users of the system (Presentation and Application Layer).

The transport layer, also known as the host-host layer, accepts data from the session layer, splits it into smaller units if needed, passes these to the network layer, and ensures that all the pieces arrive correctly at the other end. It creates a distinct network connection for each transport connection required by the session layer, or may multiplex several transport connections onto the same network connection, to reduce the cost. In all cases, the transport layer makes these network connections transparent to the session layer. This layer is a true source to

destination or end-to-end layer, so a program on the source node carries on a conversation with a similar program on the destination node, using message headers and control messages. In contrast, at lower layers the protocols are carried out by each node and its immediate neighbors (chained-layer).

The session layer represents the true user interface to the network. It establishes the logical connection (session) between users or presentation-layer processes in different nodes, and maintains and releases it at the users request. To do this it converts names to addresses, checks for access permission, type of communication (half duplex or full duplex), etc. It also provides connection services such as recovery, diagnosis and statistics (mainly for performance measurements and billing). In some networks the session and transport layers are merged into a single layer.

The presentation layer performs functions that are requested sufficiently often by the users. It is then better to have the system provide them as services, rather than allow each user to perform them in its own way. These services include any conversion of information that might be needed as it is transferred between end users. Some examples are data compaction, expansion, encryption, sending and receiving formats, convertion of data types and data representation, terminal handling and file transfer.

The application layer represents the highest level layer in the network. Its contents usually depends on the individual users, since they are the ultimate sources and destinations of information passing through a network. The application layer does not become directly involved in communication functions, and thus the need for the end user to know about internal communications procedures and protocols of the network is eliminated. The presentation layer represents the domain of the network users, while the

lower level layers represent the domain of the network designers.

III. ROUTING

A. DEFINITION

A central issue in the design of a Packet Switching Network is the selection of the routing protocols which are responsible for deciding the path to be followed by a packet as it traverses the network. The basic concern of a routing protocol is the efficiency of the information transfer. Its main goal is then to determine the optimal (best) routing policy, i.e.,the set of routes over which packets have to be transmitted, in order to optimize a well defined objective function (like delay, cost, throughput, etc.). It asumes that the integrity of this transfer is provided by other protocols (error protection, duplicate detection, security, etc).

In optimizing an objective function, as for example minimum-delay, we can use two different criteria: optimize the average system delay, or optimize the individual delay for each source-to-destination traffic requirement. Fortunately, most routing solutions obtained using these two approaches turn out to be similar (less than 1% difference), especially for large networks with uniform requirements [Ref. 6]. The system optimization approach, as we will see, is generally used in static routing problems, while the individual or user optimization concept is most common in dynamic routing problems.

The degree of difficulty of the routing problem in any network, is strongly influenced by the network topology. In a star-connected network with full duplex links, for example, only the control node must have the routing information since all traffic must go through it, and the

routing will simply be a match of the destination to the output link that connects to that node. Another simple arrangement is a single ring-connected network. If the links are full-duplex, traffic can reach any destination from any source either way around the ring, and the routing algorithm can be quite trivial, needing no routing table as main component since the shortest path can be easily computed from the destination node number. But networks are not always that simple, and more and more sophisticated routing protocols are required as the complexity of the network topology increases. Even without knowing the details of the network traffic, it is possible to make some general statements about the optimum routes, based on the network topology. One example is the Optimality Principle, which states that if node "B" is on the optimal path from node "A" to "C", then the optimal path from "B" to "C" falls along the same route.

Some of the most important packet switched communications networks in current operation are TYMNET (Public common carrier network based in the U.S.), TRANSPAC (French's Government PTT data network), ARPANET (U.S. Department of Defense Computer Network), SNA (IBM's System Network Architecture) and DNA (Digital Equipment Corporations Digital Network Architecture). The routing algorithms used in these networks turn out to be variants, in one form or another, of shortest path algorithms that route packets from source to destination over the least cost path. The specific cost criterion used differs among the networks. Some use a fixed cost for each link in the network, and usually the cost is assigned inversely proportional to the link transmission capacity, in bits per second. For a network with equal capacity links, minimization of the path cost then generates a minimum hop path. Links with high error rates or congestion may be

assigned higher costs to avoid sending too much traffic through them. Other networks attempt to estimate average packet time delay on each link, and use this to assign a link cost. The resultant source-destination path chosen, tends to be the path with minimum average time delay. Once the best paths have been determined, routing tables set at each node are used to route individual packets to the appropriate outgoing link.

Shortest path with single routes turn out not to be optimum, if the long-term average network delay time is to be minimized. In this case multiple paths, where fractions of the packets at a node are assigned to one of several outgoing links (perhaps on a probabilistic basis), provide for a closer to optimal routing. This kind of routing has not yet been used in routing schemes implemented in operating networks, although there are plans to incorporate this procedure in future routing mechanisms for the Canadian DATAPAC network [Ref. 7]. In the present study we investigate the use of a routing protocol that allows for the routing of packets over multiple paths.

B. STATIC AND DYNAMIC ROUTING

Routing protocols can be grouped into two major classes: Adaptive or Dynamic and Nonadaptive or Static. Dynamic protocols base their routing decisions on measurements or estimates of the current traffic and topology of the network, and their routing policies vary in time according to the fluctuations of these measurements. Static protocols, in contrast, make those decisions independently from the current status of the network. Their routing policies are determined a priori and are time invariant. For this reason, static protocols must be designed to provide satisfactory performance, on the average, over a range of traffic intensities.

If a dynamic protocol could manage to adapt perfectly to the changes in traffic and topology of the network, it will naturally outperform any static protocol; but traffic loads vary continuosly, and topology changes in most cases can not be predicted, so dynamic protocols require complicated computations which may slow the operation of the network. Furthermore, sending the update information through the network may itself increase the traffic. Static protocols, in contrast, are usually simple, and since their implementation is done before bringing up the network, they do not affect the traffic load. Nevertheless, static routing protocols do not react to drastic changes in traffic or topology in the network, and since most traffic is bursty by nature and equipment malfunctions are not infrequent, they may lead to congestion and misuse of some of the links.

Static Routing is widely used in network design. There we are interested in determining whether a given traffic pattern can be accommodated by the network, and if it can, what will the average delay be. Network topology is generally designed interactively by producing small changes (addition/deletion of links or reducing/increasing link capacities) and observing the effects on throughput and delay performance. This performance evaluation is accomplished using a static routing algorithm. A method that applies this process is the "Cut-Saturation" algorithm for the topological layout of packet switched networks [Ref. 8].

The degree of adaptivity of a certain routing protocol can be measured in terms of its response time, that is, the rate of change in the traffic and topology of the network which it is able to track efficiently. There is indeed a whole spectrum of sclutions to the routing problem between the two extremes mentioned (static and dynamic). They are characterized by different response times and used for

different applications. A comparative analysis of some routing alternatives that combine static and dynamic routing features has been done by Rudin [Ref. 9]. If the rate of change of conditions in a network is slow, for example, a periodically refreshed static routing (or "quasistatic") protocol, can be a reasonable solution; but if it is fast, a dynamic routing scheme may be required.

In the static (or quasistatic) routing enviroment, the time between traffic pattern changes is very much larger than the average network transmission time, so that the routing computation can be performed in the background, at a very slow rate, and without increasing the load on links or nodes much. In the dynamic routing enviroment, on the other hand, the time between traffic changes is comparable to the time required to measure such changes, compute new routes and implement them. In this case we must be concerned with the time required to arrive to an optimal solution and install it in the nodes, and the amount of overhead introduced in both link and node usage. The frequency of updating routing tables represents a compromise between the desire that as soon as a change is detected the information is immediately made known in the network, and the amount of overhead generated by the updates.

For a fixed network topology and fixed traffic pattern situation, the optimal static routing protocol represents the optimal routing scheme. For deterministic steady input then, dynamic routing protocols that use the system optimization approach must converge to the optimal static routing, that corresponds to the existing traffic pattern and network topology.

The effectiveness of different static routing protocols relative to dynamic routing protocols is not really known, except for some suggestions from simulation. For static routing, a well known lower bound for delay has been

determined by Fratta, Gerla and Kleinrock with their "Flow Deviation Method" [Ref. 10]. There is no corresponding lower bound known for dynamic routing, but in some cases, simple dynamic routing schemes give delays significantly lower than the best static routing delay for some specific network topologies. Examples can be found in YUM [Ref. 11]. Simulation studies at the British National Physical Laboratory indicate that the addition of a dynamic capability to a shortest path (static) routing rule may increase throughput by as much as 50%. [Ref. 12].

An optimal routing strategy will then use a dynamic protocol that behaves as a static one, when traffic and topology of the network warrants it, and as a normal dynamic one otherwise. Such a routing protocol should not be sensitive to small variations in the traffic, and it should allow routing through multiple paths to maximize utilization of resources and reduce packet time delay.

C. DISTRIBUTED AND CENTRALIZED ROUTING

Perhaps the most important characteristic that differentiates the numerous routing schemes that are currently used in packet-swithched communications networks is the strategy used to control the selection and implementation of the routes under different network conditions. McQuillan [Ref. 13], defines four categories of control strategies:

(1) Deterministic Control
(2) Isolated Control
(3) Distributed Control
(4) Centralized Control

Deterministic Control implies fixed routes, but nodes have precomputed tables which can be manually selected to deal with line or node outages. An example of this kind of

network is the SITA network. Isolated control strategies involve decision making at each node based only on local information. Information is not even shared with adjacent nodes, and thus, changes are not propagated out of the node. Distributed control allows each node to make its own decission on how to adapt to the changes in the network, based on information obtained locally as well as from other nodes. Furthermore, this strategy tries to propagate routing information of global importance to all the nodes. The last category, centralized control, assigns the responsibility for all routing decisions to a single node, usually called the Network Routing Center or NRC (possibly backed up by alternate nodes in case of failure). In this case, all nodes must report their status and any network changes to the NRC, as well as receive and adopt routing decisions from it. Of these four categories, the two most widely used are Distributed and Centralized Control. Some examples of networks using distributed routing are ARPANET and DATAPAC, while TYMNET uses centralized routing.

Regardless of whether centralized on distributed control is used, they both impose in the network a cost in time, bandwith, buffer space and other network resources, to move routing information from the locations where routing policies are selected to where they are enforced. In addition, if any dynamic routing scheme is used, it must overcome a problem fundamental to them, namely, the short time interval available to deliver routing information to and from the nodes before network conditions have changed so much that the information is obsolete.

For dynamic routing protocols we prefer for the decisions to be made in a distributed fashion. The reason is that using the centralized approach, the time it takes the NRC to receive information from all the nodes, calculate the routing strategy and send it back to the appropriate

nodes, can be so long that the routing decision may be out of date. In this case the dynamic routing protocol may loose its adaptiveness and make what is effectively a random change. In contrast, distributed schemes do not have as much information, since we can usually expect to have only the past information of all the network (global), the present information of the node (local), and partial information of some other node or group of nodes. The decision rules are usually simpler, and the propagation delay for the exchange of status information between neighboring nodes, can also be smaller than if sent from a centralized center. This reduces the time elapsed between the change of states and the change of the routing decision. A distributed scheme reduces the reliance on a cental node that may fail. It may also diminish the effects of node and link failure, since in a centralized protocol, the routes to be used by the notification of such failures may in fact be destroyed by them. In addition it can avoid the increase of traffic overhead in the proximity of the NRC, due to the periodic collection of status reports from all the nodes, and the distribution of routing decisions from it.

Although no mixed scheme has yet been implemented, in principle it is possible to combine routing strategies to compensate for each other's defficiencies, and therefore hope to reach reach an overall improvement of routing characteristics. An example of a routing protocol that combines centralized with distributed routing is the Delta-Routing. In this scheme the central controller sends routing tables to the individual nodes, but these are used in conjunction with local queue lengths to select the routes to be taken by individual packets. Other schemes that combine centralized with distributed routing are discussed by Chu and Shen [Ref. 14].

D. FLOW CONTROL AND CONGESTION CONTROL

In most shared communication networks, resources are dimensioned to meet less than the peak demand requirements, due to cost factors. This underdimension introduces a reduction in the usual performance, which should be acceptable to the users, plus a risk that the actual traffic load may occasionally exceed the level where acceptable performance is obtained. The traffic load level at which acceptable performance is just met, and where any increase in it will make performance unacceptable, is called the overload level. By setting this level the designer can make the risk of non-acceptable performance as small as he wants. Therefore, the overload level is the network's design load level.

In a packet switched communications network, even when the traffic level does not exceed the overload level, there is a need to control the rate at which packets flow from one node to the next and to prevent packets from arriving at the receiver at a rate faster that it can handle them. We must also make sure that the load imposed on the path between transmitter and receiver does not exceed its capacity. A mechanism to allocate resources to satisfy user demands as long as there are resources, and to settle contention when the network runs out of resources, is required. This mechanism is usually called Flow Control. A good definition of Flow-Control given by Rudin [Ref. 15], states that flow control is "a system of algorithms which are used in a network to prevent a single user or a user group from monopolizing the network resources to the detriment of other users".

Flow Control may be done between neighboring nodes (local control) or between source and destination (end-to-end control). Between neighboring nodes the flow

control mechanism must be design to ensure that the transmitter sends packets at a rate acceptable by the receiver and thus protects the receiver node against over-flowing its packet buffers and overloading its processing capacity. Consequently, it places limits on the number of packets at the buffers of the switching nodes. Between source and destination nodes, flow control must be designed to ensure that the source node does not generate packets for the destination at a rate higher than that at which the latter can accept them. So it places a limit on the number of packets belonging to a source-destination pair.

All flow control mechanisms either require the sender to stop sending at some point and wait for an explicit go-ahead, or permit the receiver to discard packets at will with impunity. Two well known flow control mechanisms are Stop-and-Wait and Sliding-Window. Some analytic models for the study of end-to-end and local flow control can be found in the literature [Ref. 16].

When the number of packets released into the network by the nodes is within its carrying capacity, they are all delivered (except for a few that may be affected by transmission errors). But when too many packets are present in part of the network, it can become impossible for packets to move, because the queues into which they should be accepted are always full, causing the network performance to degrade; this situation is called Congestion. Congestion can be brought about by several factors. If the nodes are too slow to perform the various tasks required (queuing, updating tables, etc), their queues can build up even though there is excess capacity in their output links. On the other hand even if the processing speed of the nodes is infinitely fast, queues can build up if the input traffic rate exceeds the capacity of the output lines. This can

happen for example if two or more input lines are delivering packets all of which need to be sent by the same output link. Congestion tends to feed upon itself and become worse if nothing is done to control it. If the traffic increases too far, performance may collapse completely and almost no packets be delivered.

Many factors can cause the carrying capacity of the network to be exceeded and cause congestion, so the network must be able to react and take some action to control it. Flow control techniques cannot really solve the congestion problem because traffic is bursty, so any scheme which is adjusted to restrict each user to the mean traffic rate will provide bad service when the user wants to send a burst of traffic, even though there is no congestion. On the other hand, if the flow control limit is set high enough to permit the peak traffic to get through, it has little value as a congestion control mechanism when several users send their peak traffic at once. What is really needed is a control scheme that is only triggered when the system is congested; this scheme is called Congestion Control. Congestion control is therefore the set of mechanisms whereby the network maintains input traffic within limits that are compatible with its carrying capacity.

In most packet switched networks, if a packet remains queued for too long at some node before arriving to its destination, retransmission will occur until a positive acknowledgement is received, this process creates the ability to generate traffic within the network itself. With high volumes of traffic the generation process can create a traffic volume equal to the remaining capacity of the network. When this critical point is reached all buffers are full and the network fails and remains with no throughput; that is, it cannot receive or successfully transmit a packet, so that all nodes are caught in a deadly embrace.

This extreme case of congestion is called Lock-up and is irreversible. The only way out is to purge the locked-up traffic.

The general strategy is therefore preventative in nature, and consists of controlling congestion in the first place. If infinite buffers were provided at the nodes, there would be no lost traffic and the network would only congest at infinite load, well above the overload point (the point of acceptable delay). If to the contrary buffer size were very small, the network will congest even at small traffic loads, well below the overload point. Thus there is an optimal buffer size which makes overload and congestion occur at the same traffic level. This is desirable because it is better to have a congestion control mechanism that rejects overloading traffic than to have excess buffers in which traffic may spend too much time to be useful.

Without effective congestion control, packet mean time delay will tend to infinity and throughput will tend to zero, as the network approaches the deadlock load. With effective congestion control, mean time delay and throughput will increase until they reach the saturation level and then stay the same (flatten), remaining insensitive to further load increases. The curves in Fig. 3.1 show network throughput and packet transit time delay versus offered load, for networks with and without effective congestion control.

It should be apparent that with effective congestion control, if the network throughput and packet mean transit delay flatten as offered load increases, the mean admittance delay of packets to the network by the nodes must increase. Fig. 3.2 shows a comparison between admittance delay and transit delay, for packets in the network, as offered load increases.

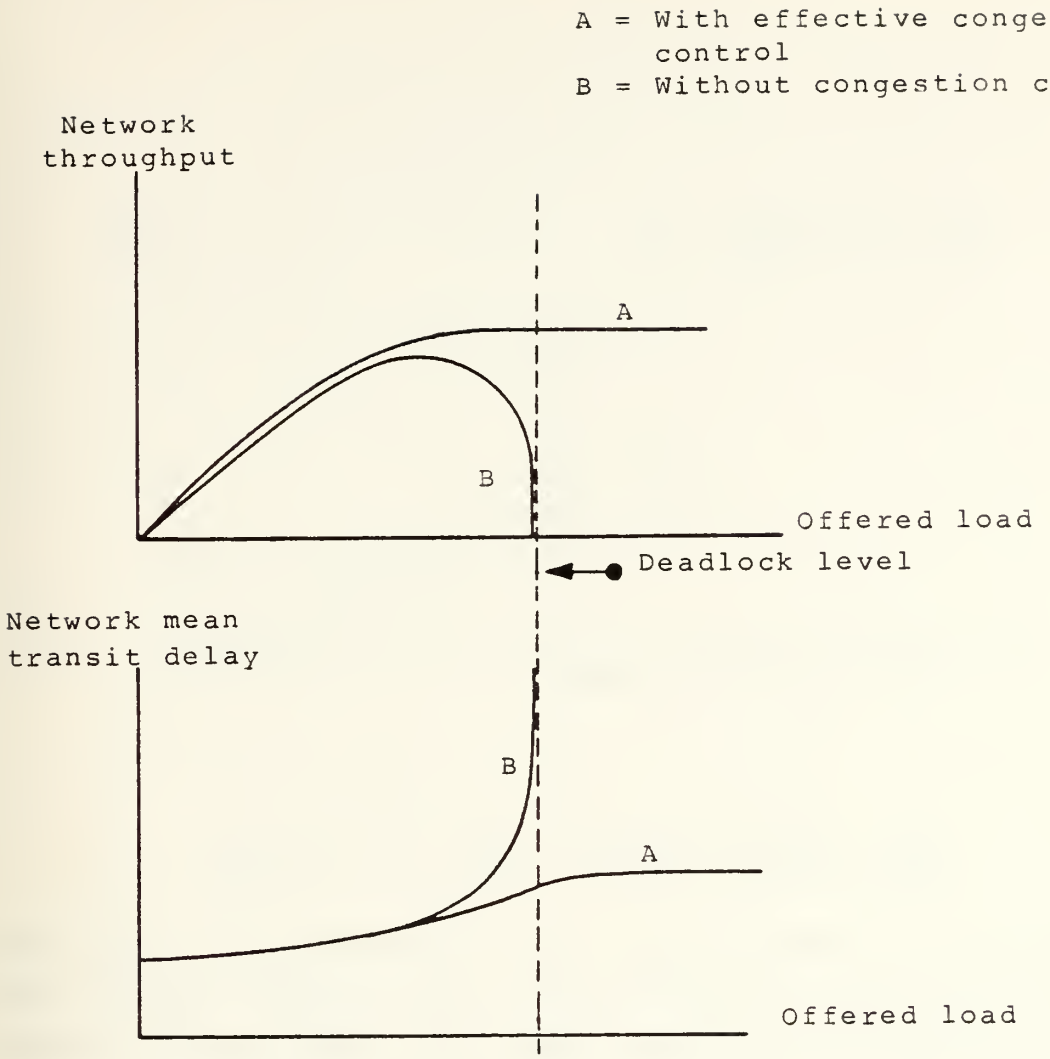

Figure 3.1 THROUGHPUT AND TRANSIT DELAY VERSUS OFFERED LOAD.

Some common types of strategies used in packet switched networks for dealing with congestion use pre-allocation of resources, choking off input (that is, requiring the node sender to reduce its traffic by sending it a Choke-Packet), packet discarding, and restricting the total number of packets allowed in the network (Isarithmic control). Their effectiveness and performance are analyzed extensively by

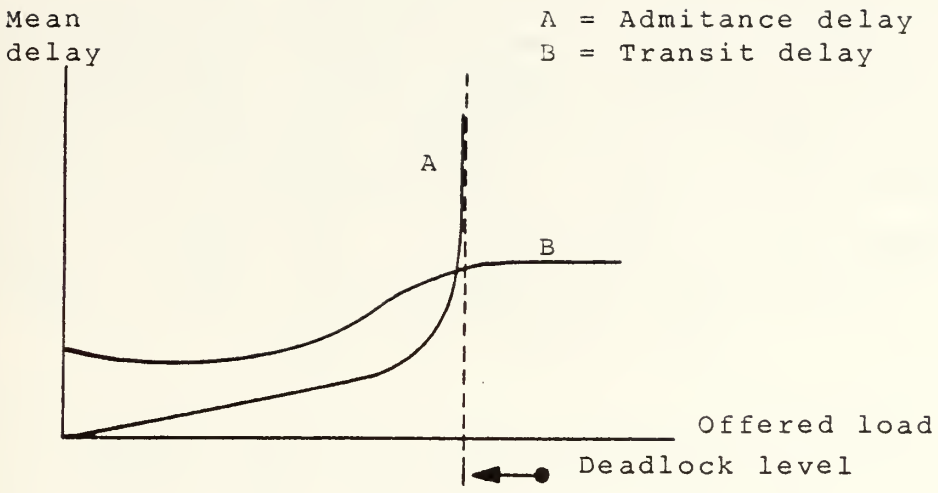

Figure 3.2 DELAY VERSUS OFFERED LOAD.

SCHWARTZ and SAAD [Ref. 17], Ireland [Ref. 18] and Davies [Ref. 19]. Since poor routing algorithms often lead to congestion problems, and local congestion often requires at least temporary modification of routing protocols, the routing problem cannot be completely divorced from that of congestion control. However, in this thesis we concentrate on the routing protocols under the assumption that the better they are, the less congestion is likely to occur in the network.

E. ROUTING FRACTIONS

Under the appropriate assumptions the optimal routing problem can be formulated as a nonlinear multicomodity flow problem, so that mathematical programming techniques can be used to solve it. All the routing protocols mentioned so far are based on the assumption of the convexity of the objective function. A very popular analytical model of data communications networks, which relates the packet time delay to the flows and capacities of the links, was introduced by Kleinrock in [Ref. 20]. In this model, the steady state time delay in each link is calculated explicity as:

$$Tij = 1 / (Cij - Fij)$$

where:

- i : Is the node origin of Link (i,j).
- j : Is the node termination of Link (i,j).
- Tij: Is the expected delay per message experienced by messages using Link (i,j).
- Cij: Is the capacity of Link (i,j) in messages per second.
- Fij: Is the data flow rate in Link (i,j) in messages per second.

The routing assignment is then selected to minimize the expected weighted delay "D", of messages traversing the network, where:

$$D = \sum_{(i,j)} (Fij \times Tij)$$

This analysis is based on the assumptions that traffic in each link can be modeled as Poisson message arrivals with independent exponentially distributed message lengths, and that queueing delays are the only nonnegligible source of delay in the network. The first assumption is the result of Kleinrock's famous "Independence Assumption", that messages

loose their indentity at each node and are assigned new independent lengths. The second assumption, generalized by Kleinrock in [Ref. 21], accounts for overhead and propagation delays. Kleinrock's model is appropriate for static and quasi-static situations but less appropriate for dynamic strategies.

A generalization of this model, where the contents of the queues at the nodes are viewed as continous quantities, rather than as an integer number of messages or bits, is proposed by Segall in [Ref. 22]. The continuous nature of his model is justified by the fact that the effect of any single message on the total system performance should be minimal. Using this continuous model the routing problem can be formulated as a linear optimal control problem. Solution to this problem has been approached via a feedback form, obtained by means of Pontryagin's minimum principle, and dynamic programming. Another solution involves replacing some constraints with penalty functions, and uses this formulation to investigate how to minimize the average message delay, while disposing of whatever backlogs may exist in the network at any particular point in time. A comprehensive study of the feedback sclution has been presented by Moss in [Ref. 23], and solution has been obtained for the case in which all messages have the same destination and the inputs are constant in time. But the approach runs into difficulties when the general case is considered and no solution has yet been obtained for it.

A different approach proposed by Wozencraft [Ref. 24], involves changing the objective function while keeping the same continuous model. Instead of trying to minimize the average delay, the idea is to minimize the maximum delay. This minimax approach has been extensively researched by Ros in [Ref. 25]. The objective is then to minimize the maximal saturation ratio (Fij / Cij) in the network. Subsequently

the next maximal saturation ratio is minimized. Iteration is continued in this way until a unique solution to the corresponding problem is found or we have exhausted all links. At this point we can generate a finite set of fractions (we will call them Routing Fractions) that represent the proportion of traffic from node "A" (any node) to node "B" (destination node) that must be routed through link "C" (any outgoing link of node "A"). We will call this procedure: "Successive Saturation".

An alternate solution which requires less computational effort consists of doing the first iteration of the Successive Saturation method mentioned before, using its results to scale the capacity of the links in the network, and then finding the set of flows that maximize the sum of the exess capacity (slack) of all links. At this point we have found a finite set of Routing Fractions, as defined before. We will call this procedure: "Maximum Slack". Both solutions use linear programming techniques and therefore enjoy the tremendous computational benefits that they offer.

This thesis is part of a bigger effort to analyze the use of Routing Fractions in the routing protocols of a packet switched communications network. As such it concentrates on the study of their behavior in a static multiple path routing strategy. Nevertheless, the model developed as well as the computer simulation program contain provisions for future experimentation on their use in a dynamic routing strategy. These provisions allow the addition of a computer program that calculates the values of the Routing Fractions (which is the matter of another thesis being undertaken at the present time). This program could then be called upon to recalculate the values of the routing fractions, when the traffic load or topology changes in the network warrant it.

F. LOOP AVOIDANCE

An important characteristic that any routing protocol should have is to be "loop-free". Loop-freedom defines a per destination partial ordering of the nodes in the network, which is used by the routing protocols for the propagation of packets through the network, to prevent them from looping. A loop-flow exists in the routing protocol when a packet can potentially loop, that is, it can arrive to the same node for the second time in its way to its destination. A given packet may be trapped in such a loop for a significant amount of time. If a large number of packets start looping they can cause congestion (since retransmission will occur until a positive acknowledgement is received) which begins at the locus of the loop and can spread throughout the network. A loop-flow may exist even if no individual packet ever loops.

The main reason for a routing protocol to be loop-free, is the reduction in packet time delay. Other important reasons include the simplification of higher level protocols, and to prevent some potential deadlocks.

Most rules and procedures used to develop loop-free routing schemes, are extensions of techniques used in network graph theory and flow pattern analysis. The following are some interesting examples of routing principles derived from these techniques.

If we call the destination node "Sink", and if node "A is on a path from node "B" to the sink, we say "A" is downstream from "B", and "B" is upstream from "A". In a loop-free routing protocol, no node can be both upstream and downstream from any other node, for a given destination. If "T" is a set of links that form a tree in the network graph, and "F" is a set of links (not in "T") going into node "A", then the set (T + F) is loop-free since the links going into

node "A" cannot be in a loop. Furthermore the set (T + F - (ALL LINKS NOT IN F)) is a tree. Sums of loop-free tree flows are not always loop-free as might be expected [Ref. 26].

The set of optimal routes from all sources to a given destination form a tree routed at the destination (such tree is called the Sink Tree). Therefore, it will not contain any loop-flows, and each packet will be delivered to its destination within a finite and bounded number of hops. If a fixed routing is contructed by choosing a maximal tree in the network graph (that is using only links directed towards the sink), then the resulting flow is nonnegative and loop-free, and the set of all flows determined this way is clearly convex and loop-free.

In general, no optimal routing protocol can have loop-flows, a fact which will tend to minimize congestion. Some routing algorythms cannot be said to be loop-free but if the small number of loops which they do form do not persist (transient), and do not lead to congestion or to a significant increase in average network delay, they may be accepted as efficient. An example of this case is the new ARPANET routing scheme SPF (Shortest-path-first) [Ref. 27], where a small amount of transient looping occurs while the network is adapting to a routing change.

An algorithm that does not permit looping does not necessarily result in lower delay, higher throughput, or less congestion than an algorithm which does. However research by Gallager [Ref. 28], has proven that the paths in a minimum distance (optimum) solution, for a network with link distances greater than zero is loop-free.

IV. SIMULATION OF A PACKET SWITCHED COMMUNICATIONS NETWORK

A. MODEL CHARACTERISTICS

In the analysis of the routing protocols of a packet switched network, simulation has proven to be a powerful tool to investigate their performance and carry out comparative studies of the different strategies. In some cases, such as with distributed dynamic routing, simulation has been relied on almost exclusively because of the time vaying behavior of the set of interactive queues, whose theoretical study is still in its infancy.

In simulation the network and its protocols are modeled in terms of a computer program. Nevertheless, it must not be an emulation seeking to duplicate every small detail of the network operation. A simulation model that tries to cover every detail is an extremely expensive, wasteful, and slow-running operation. As a general rule one should only simulate the system features that are relevant. Therefore the simulation of a routing algorithm must describe the nodal queue handling procedures, and the routing process itself, in full detail.

The model we chose is that of a set of nodes connected by unidirectional communication links. The number of outgoing or incoming links is not limited by the model; this allows for the configuration of the network in almost any topology desired. Each link has a buffer (we will call it a queue) where packets are stored if the link is busy. Transmission of packets from the queue of each link is done in a FIFO (First in first out) fashion. Nevertheless, some packet-pricritizing scheme can be easily added to the model, without affecting its structure.

Our model assumes noise-free links, that is, packets arrive at their destination without any errors and no packet loss is experienced during transmission. Similarly, queueing delays and transmission time are assumed to be the only nonnegligible sources of delays in the network.

Message traffic in a packet switched network can be modeled as a Poisson process, with the length of the messages geometrically distributed. Accordingly, our model uses a message generator that generates poisson distributed messages (by using exponentially distributed interarrival times), and the number of packets for each message (message length) is generated from a geometric distribution.

Traffic load is another parameter that can be set to any desired level in our model, and two ways of doing it are provided. The first accomplishes this by setting the average number of new messages for the network per second. The inverse of this number is then used as the mean of the exponential distribution used to generate the message interarrival times as mentioned before. The second does it by setting the average number of packets per message. This number is then used as the mean of the geometric distribution from which the length of each message (in packets) is generated. Both numbers are read as inputs by the model so they can be set to any desired value.

The model accepts as inputs the probability of each node being the source of a message, and the conditional probability of each node being the destination given a source node. With these two sets of inputs, it calculates the probability of each possible combination of nodes (node-pair) being source and destination of a message. When a message is generated, the model picks a node-pair randomly and assigns them as source and destination of the message respectively.

As indicated in chapter three, the Routing Fractions represent the proportion of traffic from node "A" (any node) to node "B" (destination node) that must be routed through link "C" (any outgoing link of node "A"). This nature of the Routing Fractions make them readily available for their use in a multiple path routing scheme.

The routing protocol procedures of the model are contained in a separate module (see routine 'PICK.BEST.LINK' in the program description section), to allow them to be easily changed if required. Values of the routing fractions corresponding to each link are read by the model, and used by the nodes to select the outgoing link by which to route each packet. This selection is done by picking a routing fraction probabilisticly, and using the link that corresponds to it as the selected outgoing link. This procedure, as mentioned before, allows for multiple paths in the routing of traffic destined to any node.

Since our study concentrates on their use in a static routing scheme, the values of the routing fractions do not change throughout the simulation run. Nevertheless, the model allows for the addition of a Routing Fraction recalculation procedure, which could be executed when the traffic load or topology changes in the network require it. This addition will change the model's static routing scheme into a dynamic one.

In order to make the model more flexible in adapting to different testing conditions, we allow the network to be configured to provide either Datagram or Virtual Circuit services. When in the Datagram Service Configuration, the route in terms of links to be traversed by each packet (on its way from the source node to the destination node), is determined sequentially. As the packet arrives to each node, the next outgoing link to be traversed by it is determined by the model. This procedure is then repeated at each node

until the packet arrives to its destination node. There, data from it is collected by the model for the statistical analysis of the network behavior, and then destroyed by the model. In this way, each individual packet will traverse a route specially determined for it at each node, and no direct relationship exists between this route and the routes of the other packets of the same message. Consequently, in this procedure packets of the same message can (and most often do) follow different paths through the network on their way to their destination. This as we saw before, complies with the characteristics of Datagram Service.

In the Virtual Circuit service configuration, in contrast, the complete route to be followed by a message is determined at the moment of its creation. This route will then be followed by all the packets belonging to that message. No route change is therefore experienced by any packet after it leaves the source node. When a packet arrives at a node, the model checks its predetermined route and sends the packet to the corresponding outgoing link. This procedure is repeated at every node until the packet arrives to its destination, where data from it is collected (for the same statistical purposes as in datagram) and the packet destroyed by the model.

As it is in most packet switched networks in operation, in our model the size of all the data packets is the same, but it can be defined to be any number of bits. In order to do this the model accepts this size as an input, and fixes the size of all the packets to it. If there were a requirement to allow packets to be of different size, a mechanism to generate these sizes could be easily added to the model.

Another important parameter of our network model is the link capacity, i.e. the data carrying speed of each link, in bits per second. Our model reads the capacity of each link

and then uses it to calculate the time required for a packet to traverse that link (transmission time). This transmission time is further used to schedule the arrival of the packet at the next node. The model can therefore accept any value of capacity for each individual link, as long as it is in bits per second.

As mentioned before, each link has a buffer or queue to store packets waiting for transmission. In our model we have chosen not to limit the size of these queues, in order to allow the study of the behavior of the routing fractions without introducing any other process that could affect the results of the analysis.

Most of the parameters of the model can be changed very easily, to allow the simulation of different testing conditions, as we will see. As an example the number of nodes and links do not have a theoretical limit, but must be set before the simulation starts, and remain fixed for the duration of the run. Nevertheless, there are some indirect ways of changing the topology of the network during the run, such as setting the corresponding routing fractions of any link to zero. This will prevent anymore traffic from going into it. In this way we can represent a link failure.

B. PARAMETERS OF THE PACKET SWITCHED NETWORK

The definition of the different variables whose values must be read as inputs by the program is the following:

1) N.NODE : Number of nodes of the network, integer number.
2) N.LINK : Number of links of the network, integer number.
3) CONNECTED : Integer two-dimensional matrix that indicates the way the links are connected in the network. Each row represents a link and each column a node. So position CONNECTED(A,B) must have a value of 1 if link 'A' starts in node 'B', a value of -1 if

link 'A' ends in node 'B' and a value of 0 elsewere.

4) PR.ORIG : Real 1-dimensional matrix, each row represents a node and contains the probability of a message being originated in that node.

5) PR.DEST : Real two-dimensional matrix, each row represents an origin node and each column a destination node. So position pr.dest(A,B) contains the probability that a message that was originated in node 'A' has as its destination node 'B'.

6) ROUT.FRAC : Real two-dimensional matrix, each row represents a link and each column a node. So for position ROUT.FRAC(A,B), it contains the proportion of the traffic destined to node 'B', that should be sent by link 'A' i.e. routing fraction of link 'A' for node 'B'. If no traffic for node 'B' can be sent by link 'A' its value should be zero. For our study (static scheme), these values do not change during the simulation run. For the future use of this program in a dynamic routing scheme, these will only be the initial values of the routing fractions. During the simulation run, their values should be recalculated every time the conditions in the network so require (see event 'NEW.ROUT.FRAC' in section "D" of this chapter).

7) LNK.CAPACITY : Integer 1-dimensional matrix, each row represents a link and contains the capacity of that link in bits per second.

8) TIME.LIMIT : Real number that represents the length of time the simulation run will last, in seconds.

9) REPT.TIME : Real number that represents the interval of time between 'NET.REPORT' events, in seconds.

10) UP.TIME : Real number that represents the interval of time between 'NEW.ROUT.FRAC' events, in seconds.

11) COLL.INT : Real number that represents the interval of time between 'DATA.COLLECTION' events, in seconds.

12) LAP.TIME : Real number that represents the interval of time between 'LAP.TOTALS.RESET' events, in seconds (see event 'LAP.TOTALS.RESET' in section "D" of this chapter).

13) LENGTH.PKT : Integer number that represents the length of a packet in bits.

14) AVG.MPS.NET : Real number that represents the average number of messages per second the network will generate.

15) AVG.PKTS.MSG : Real number that represents the average number of packets-per-message the network will generate.

16) PRNT : Integer number that represents the printing condition for the run. Depending on the value set it can print the following:

    0 ==> Initial data, network topology, net reports, data collection messages, period reports, collect period data messages, restart period totals messages, lap totals reset messages, destruction message, and end of simulation message.

    1 ==> All in 0 plus trace of all packets.

    2 ==> All in 0 and 1 plus information of all links every time it performs a 'NEW.MSG' or an 'END.XMT' event.

17) MO.DE : Integer number that represents the configuration of the network for the simulation. It can take the following values:

    1 ==> Datagram service network

    2 ==> Virtual Circuit service network

The topology of the network used for our simulation is shown in figure 4-1. It consists of a set of thirteen nodes connected by sixty unidirectional links. To simplify the drawing each line interconnecting two nodes in Figure 4.1 represents two links, one in each direction. Each link in our model is assigned a number for its identification. Appendix B contains a list of the link numbers with their corresponding origin and termination nodes. As mentioned before, each link owns a queue (or buffer) where packets are stored temporarily while waiting for their transmission if the link is busy. These queues have no limit in their size.

The link capacity (the rate at which data is carried through the link measured in bits per second) of all links was set to 20,000 bits per second, which is the transmission rate that modern modems allow over dedicated lines. The packet length used was 1000 bits, which is approximately the maximum packet size allowed in ARPANET (1008 bits). Transmission time is calculated by the model every time it starts to transmit a packet through a link. For this it uses the capacity of the link involved, and the packet length.

The probability of a node pair to be selected as source and destination of a message, was initially set to be equal for all nodes. Later a non-uniform distribution was also used to compare the behavior of the routing scheme in balanced and unbalanced traffic situations.

The mean number of messages per second for the network and mean packets per message where set to different values to analize the response of the network to different traffic loads.

Two sets of routing fractions were used. The first set was calculated by the successive saturation method and the second set was calculated using the maximum slack approach. as explained in chapter III. Both sets where tested under the same network and traffic conditions.

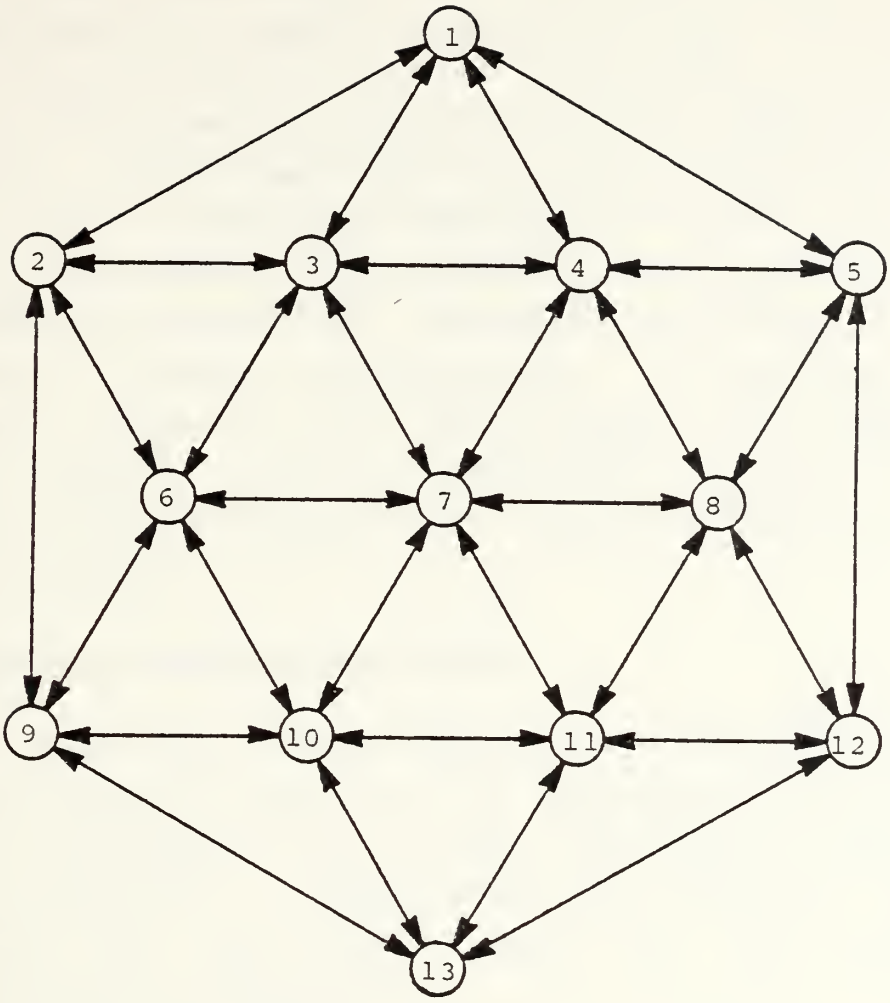

Figure 4.1 NETWORK TOPOLOGY.

C. COMPUTER LANGUAGE

The simulation of our packet switched network model was done using the SIMSCRIPT II.5 Computer Language. This language was selected because it provides an excellent means for discrete-event simulation. Another important characteristic of this language is that it allows the use of Fortran subroutines. This feature makes it possible to take

advantage of existing subroutines or programs, rather than to waste time reprograming them in SIMSCRIPT II.5.

SIMSCRIPT II.5 language uses english-like statements and allows for the insertion of some extra words, that make it easier to read (even by someone with little practice in this language). These characteristics make a program written in Simscript II.5 almost self-documented and only some short comments are needed. Nevertheless, care was taken to make the program as modular and structured as possible, to make it easier to change and understand. In addition to the simulation program, two other programs where written in Fortran language for the display and graphing of data collected from a simulation run, using the Disspla graphics software package.

D. SIMULATION PROGRAM DESCRIPTION

In writing the program one of the main objectives, as mentioned before, was to make it flexible so as to adapt to possible variations in the network topology or requirements of the simulation. Consequently the values of the input parameters can be set to adapt to most simulation requirements, as seen before.

The program is divided in three major parts, the first part is called the PREAMBLE, the second is called the MAIN, and the third contains all the Subroutines and Events.

The preamble contains mode and dimension definitions of Global Variables, declaration of Events and Event priorities, and definition of Statistical Variables and program Functions.

The main program is the driver of the simulation, it calls and schedules the initializing Routines and Events, and starts and ends the simulation.

The modularity in this program is mainly accomplished by dividing the rest of the code into subroutines and Events. The main difference between a Subroutine and an Event, is that a Subroutine is called upon to perform a function (or procedure) by another Subroutine or Event, whereas an Event is scheduled to occur (perform its function) at a certain point in time. While Subroutines return control to their caller an Event returns control to the timing mechanism of the SIMSCRIPT II.5 system (we will call it the Timing Routine).

The different Subroutines and Events contained in the program, and the functions they perform, are the following:

- ROUTINE NETWORK.CONSTRUCTION

  This routine reads the number of nodes and links, and constructs the network based on those parameters.

- ROUTINE INITIALIZATION

  This routine reads the values of the initial conditions chosen by the user and initializes all Global Variables for the simulation run. It then finds the probability of each possible pair of nodes being source and destination of a message, by using the input variables 'PR.ORIG' and 'PR.DEST' using probability's General Multiplication rule. After that it stores these values and the corresponding node names in the matrix 'PAIRS(A,B,C)', where "A" is the probability of the pair, "B" the source node, and "C" the destination node.

- ROUTINE PRINT.INIT.CONDITIONS

  This routine prints the initial setup of the network, and values of the initial conditions chosen for the simulation run.

- ROUTINE SELECT.NODES

  This routine selects a node-pair by picking a random number uniformly distributed from 0.0 to 1.0, and comparing it with all the node-pair probabilities listed

in an accummulative fashion. The node pair corresponding to the probability range where the random number falls, is selected. It then assigns the number correspondig to the row of the matrix 'PAIRS' that contains the node-pair selected, to the variable 'LOW' and returns this value.

- ROUTINE TO PICK.BEST.LINK GIVEN NO.DE, DE.ST

  This routine selects the best outgoing link, given the node where the packet is (NO.DE), and the packet destination (DE.ST). It first finds the value of the routing fraction of each outgoing link of 'NO.DE' that corresponds to the destination node 'DE.ST'. Selection is then done by picking a random number uniformly distributed from 0.0 to 1.0 and matching it to the values of the routing fractions found, listed in an accummulative fashion. The routing fraction that corresponds to the range where the number falls, determines the link to be used. The name of the link chosen is returned as the value of the variable 'BST.PCK'.

- ROUTINE TO SET.VIRTUAL.CIRCUIT GIVEN START.NODE AND END.NODE YIELDING 'Y' AND 'BE.LI'

  This routine finds the Virtual Circuit for a message, given its origin node (START.NODE) and destination node (END.NODE). To do this it calls the routine 'PICK.BEST.LINK' with the values of 'START.NODE' and 'END.NODE', and sets a counter to 1. When the value of 'BST.PCK' is received back, it compares the termination-node of that link with 'END.NODE'. If they do not coincide, it calls 'PICK.BEST.LINK' again, but this time with the name of the termination-node of 'BST.PCK' and 'END.NODE'. It then increases the counter to 2. The process is repeated until the termination node of 'BST.PCK', and with 'END.NODE' are the same. The

number of links in the Virtual Circuit is then returned as the value of the variable 'Y', and the name of the succesive links on the Virtual Circuit, as the values of the array 'BE.LI'.

- ROUTINE PERIOD.REPORT

  This routine calculates and prints the status and statistical data of the network, collected by the system during a period. The starting time of a period is taken from the variable 'LAST.PERIOD' (which is reset every time this routine is executed), and the ending time from 'TIME.V'. The routine 'RESTART.PERIOD.TOTALS', reinitializes all the totals and variables used in each period, so status and statistical data collected during each period is independent of the rest.

- ROUTINE COLLECT.PERIOD.DATA

  This routine calculates and outputs some period data to 'UNIT 9', for further graphing and/or analysis. The starting time of a period is taken from the variable 'LAST.PERIOD' (which is reset every time this routine is executed), and the ending time from 'TIME.V'. The routine 'RESTART.PERIOD.TOTALS', reinitializes all the totals and variables used in each period, so data collected during each period is independent of the rest. 'Unit 9' must be defined as one of the system units (printer, tape, mass-storage, etc.), before the program can run.

- ROUTINE RESTART.PERIOD.TOTALS

  This routine reinitializes the totals and global variables used in a period, and sets the period starting time, every 'UP.TIME' seconds.

- EVENT NEW.MSG

  This event calls the routine 'SELECT.NODES' and using the value of 'LOW' returned by it, finds the origin and destination nodes for the message from the corresponding

row of the matrix 'PAIRS'. Once this is done the event picks a geometrically distributed random number, following the procedure indicated in appendix A, and uses it as the number of packets for this message.

If 'MO.DE = 1' has been selected, the event calls 'PICK.BEST.LINK' (giving the source and destination nodes of the message), to find the best outgoing link (from the source node). It then finds the number of packets in the Propagation Queue (queue where packets are stored for a period equal to their transmission-time over that link, to simulate that they are been transmitted) and the Status (condition that indicates if a packet is being transmitted by the link) of this link, and saves this information for further use. After doing this, it generates the packets indicated by 'NUM.PKTS', and for each one, it creates a record with the name of the source node and files it in the 'TRIP.RECORD' of the packet. The event then files the packets in the queue of the outgoing link selected.

If 'MO.DE = 2' has been selected, the event calls 'SET.VIRTUAL.CIRCUIT' (giving the source and destination nodes of the message), to find the Virtual Circuit for the packets of the message. It then uses the first link of 'BE.LI', as the the best outgoing link (from the source node). The event then finds the number of packets in the Propagation Queue and the Status of this link, and saves this information for further use. After this it creates each of the packets indicated by 'NUM.PKTS', records the rest of their Virtual Circuit in their 'TRIP.RECORD', and files them in the queue of the link selected.

When the process is completed, for any 'MO.DE' selected, the event recalls the values of the Status and Propagation Queue size of the link selected, and only if

they were both zero (which means no packet is currently being transmitted by it or link is idle), schedules an event START.XMT for this link to be executed "NOW". It is important to mention that events scheduled to be executed NOW, within an event, are executed as soon as the event returns control to the timing routine. They preceed events having the same event-time (time at which they must be executed), that may have been scheduled before.

The event NEW.MSG then picks a random number exponentially distributed as the new message interarrival-time, and schedules another event NEW.MSG at that time from now, if there is enough time before the end of the run. If there is not enough time left it does not scheduled it. After this the event returns control to the Timing Routine.

- EVENT START.XMT GIVEN LI.NK

  This event removes the first packet from the queue of 'LI.NK' and finds out its destination node. If this node coincides with the termination-node of 'LI.NK' it schedules an event 'ARRIVAL' for this link, in transmission-time seconds from now. If they do not coincide, it schedules an event 'XMT.END' for this link, in transmission-time seconds from now. After scheduling one of these events, it files the packet in the Propagation Queue of 'LI.NK' and returns control to the Timing Routine.

- EVENT END.XMT GIVEN XMT.LINK

  This event removes the packet from the propagation queue of 'XMT.LINK'. If 'MO.DE = 1' has been selected, it checks if the termination-node of 'XMT.LINK' is equal to any of the records of the packet's 'TRIP.RECORD'. If so, the packet has looped, and the event collects information about it, destroys the packet, and returns

control to the timing routine. If not, the event calls 'PICK.BEST.LINK' with the termination-node of 'XMT.LINK' and the destination node of the packet to find the new best outgoing link. It then finds the number of packets in the Propagation Queue and Status of this link, and saves this information for further use. After doing this, it creates a record with the name of the termination-node of 'XMT.LINK', and files it in the 'TRIP.RECORD' of the packet. The event then files the packet in the queue of the outgoing link selected.

If 'MO.DE = 2' has been selected, the event reads the next link of the packet's Virtual Circuit from its 'TRIP.RECORD'. It then finds the number of packets in the Propagation Queue and the Status of this link, and saves this information for further use. After this, it files the packet in the queue of that link.

When the process is completed for any 'MO.DE' selected, except for the case when the packet looped in 'MO.DE = 1' (in this case control has already been returned to the timing routine, and no further action is taken by the event), the event recalls the values of the Status and Propagation Queue size of the link selected, and only if they were both zero, which means no packet is currently being transmitted by it (or link is idle), schedules an event START.XMT for this link to be executed "NOW". Finally the event returns control to the Timing Routine.

- EVENT ARRIVAL GIVEN ARR.LNK

  This event removes the first packet from the Propagation Queue of 'ARR.LNK', records information about it and destroys the packet.

- EVENT NEW.ROUT.FRAC

  This event was added to allow for the further use of this program, to investigate the behavior of the Routing

Fractions in a dynamic routing scheme. This can be accomplished by inserting in this event, or calling from it, a subroutine to recalculate the Routing Fractions. To aid in this recalculation of the Routing Fractions, this event collects information of queue length, utilization per period (percentage of time the link is busy per period), and utilization per lap (percentage of time the link is busy per lap), of every link.

The new subroutine can then use the information collected by this event, and any of the global variables of the program like 'LNK.CAPACITY' and 'CONNECTED', to find the new routing fractions. By setting the value of the variable 'UP.TIME', the event 'NEW.ROUT.FRAC' can be executed at any time during the simulation run, when the conditions in the network require it.

- EVENT LAP.TOTALS.RESET

This event reinitializes lap totals of the simulation, every 'LAP.TIME' seconds. The purpose of the lap totals is to allow the simulation to keep statistics of link utilization for arbitrary periods (laps) to be selected by the user by setting the value of the variable 'LAP.TIME'. The ability to calculate link utilization per lap was added to allow its use by the event 'NEW.ROUT.FRAC' when the dynamic case is analized.

- EVENT NET.REPORT

This event calculates and prints status and statistical data of the simulation, up to the time it is performed. Since variables and totals involved are not reset during the run, every time this event is performed its calculations are done for the period from time zero up to the time of execution. This event is executed every 'REPT.TIME' seconds.

- EVENT DATA.COLLECTION

This event calculates and outputs some simulation data to 'UNIT 8' for further graphing and/or analysis. Since variables and totals involved are not reset during the run, every time this event is performed its calculations are done for the period from time zero up to the time of execution. 'Unit 8' must be defined as one of the system units (printer, tape, mass-storage, etc.), before the program can run.

- EVENT DESTRUCTION

This event cancels all events that remain scheduled for execution in the Timing Routine. So after execution of this event, control of the program returns to statement following 'START SIMULATION' in the 'MAIN' (or driver of the program), and the simulation can be ended.

E. SIMULATION OUTPUTS DESCRIPTION

The main purpose of any simulation is to provide a means to observe and measure different parameters of interest, of a process that resembles the actual behavior of a specific system. Thus an evaluation of its performance and other important characteristics can be obtained, without the need for a physical implementation of the system. To accomplish this, five modules of our program where designed to collect, calculate and output information about the simulation run. They are:

-Routine 'PRINT.INIT.CONDITIONS'
-Routine 'PERIOD.REPORT'
-Routine 'COLLECT.PERIOD.DATA'
-Event 'NET.REPORT'
-Event 'DATA.COLLECTION'

The following subsections describe the contents of outputs generated by these modules, and include a brief explanation of the names used in them (which must be complemented by information given in previous sections of this chapter).

In addition to these modules, the program of Appendix F was written to allow the graphing of data collected by the event 'DATA.COLLECTION' using the Disspla Graphics Software package. Similarly, the program of Appendix G allows graphing of data collected by routine 'COLLECT.PERIOD.DATA'. A sample of some of the plots these programs can produce is shown in Appendix D.

1. INITIAL REPORT

The routine 'PRINT.INITIAL.CONDITIONS' performs two functions. First, it prints a report of the network setup and initial conditions of the simulation (we will call it "Initial Report"). Second, it outputs most of this information to 'Unit 8' and 'Unit 9', for its further use in the graphing and/or analysis of data collected by Event 'DATA.COLLECTION' and Routine 'COLLECT.PERIOD.DATA', respectively. Units 8 and 9 can be any unit of the computer system used, previously defined by the user.

The report printed by this routine is divided in four sections, which contain the following information:

a) A list of the following variables that control the simulation indicating their values read by the program:

- NUMBER OF NODES

  Number of nodes of the network, as indicated by the input variable 'N.NODE'.

- NUMBER OF LINKS

  Number of links of the network, as indicated by the input variable 'N.LINK'.

- DURATION OF SIMULATION

  Time in seconds when the run will stop, as indicated by the input variable 'TIME.LIMIT'.

- REPORT GENERATION INTERVAL

  Time interval between 'NET.REPORT' events in seconds, as indicated by the input variable 'REPT.TIME'.

- ROUTING UPDATE INTERVAL

  Time interval between 'NEW.ROUT.FRAC' events in seconds, as indicated by the input variable 'UP.TIME'. This is a provision for the future use of the program in the analyisis of the dynamic case.

- DATA COLLECTION INTERVAL

  Time interval between 'DATA.COLLECTION' events in seconds, as indicated by the input variable 'COLL.INT'.

- LAP INTERVAL

  Time interval between 'LAP.TOTALS.RESET' events in seconds, as indicated by the input variable 'LAP.TIME'.

- PACKET LENGTH

  Length of packets in bits, as indicated by the input variable 'LENGTH.PKT'.

- AVG. MESSAGES PER SECOND FOR NETWORK

  Average number of messages-per-second for the network, as indicated by the input variable 'AVG.MPS.NET'.

- AVG. PACKETS PER MESSAGE FOR NETWORK

  Average number of packets-per-message for the network, as indicated by the input variable 'AVG.PKTS.MSG'.

- PRINT CONDITION

  Print condition for the run, as indicated by the input variable 'PRNT'.

- MODE SELECTED

  Type of service to be provided by the network, as indicated by the input variable 'MO.DE'.

b) A list of all the links in the network including their name, node origin, node destination, and capacity (in bits per second), as indicated by the input variables 'CONNECTED', and 'LNK.CAPACITY'.

c) A list of the Routing Fractions that have non-zero values, including the nodes to which they correspond, as indicated by the input variable 'ROUT.FRAC'.

d) A list of the node pairs that have non-zero probability of being selected as origin and destination of a message, as calculated by the program from the values of the input variables 'PR.ORIG', and 'PR.DEST'.

Data output by this routine to 'Unit 8' of the system includes the following:

- NUMBER OF NODES (same as Initial Report).
- NUMBER OF LINKS (same as Initial Report).
- MODE SELECTED (same as Initial Report).
- PACKET LENGTH (same as Initial Report).
- AVG. MESSAGES PER SECOND FOR NETWORK (same as Initial Report).
- DURATION OF SIMULATION (same as Initial Report).
- ROUTING UPDATE INTERVAL (same as Initial Report).
- LAP INTERVAL (same as Initial Report).
- DATA COLLECTION INTERVAL (same as Initial Report).
- AVG. PACKETS PER MESSAGE FOR NETWORK (same as Initial Report).

Data output by this routine to 'Unit 9' of the system, includes the same information indicated for 'Unit 8', except for "DATA COLLECTION INTERVAL" which is not included.

2. PERIOD REPORT

The routine 'PERIOD.REPORT' collects status and statistical information during a period of the simulation, and prints it in a report-like format (we will call it "Period Report"). As mentioned before, information of each period is independant from the rest, since the routine 'RESTART.PERIOD.TOTALS' reinitializes all the variables and totals involved. This report is divided in the following three sections:

a) A list of all the links in the network (including their origin and destination nodes for ease of identification), indicating their period utilization (proportion of time being involved in transmission or "busy"); the mean, maximum value, and standard deviation of their "queue size" for the period; and the present size of their queue and propagation queue.

b) A list of the status and statistics of the network, which includes the following information:

- AVERAGE LINK UTILIZATION

  Average calculated using the "link utilization" values of all links, for the period. Its worth noting that a value of 'MAXIMUM LINK UTILIZATION' is not included as a periodical statistic, because for any useful analysis the traffic load used will cause this quantity to be 1.0 for most periods, and therefore it loses its significance.

- AVERAGE QUEUE SIZE

  Average calculated using the "mean queue size" of all links (in packets), for the period.

- MAXIMUM QUEUE SIZE

  Is the largest value observed, of the "queue size" of all links (in packets), during the period.

- STD.DEV. OF QUEUE SIZE

  Is the average value of the "standard deviation" of the "queue size" of all links, for the period.

- NUMBER OF MESSAGES GENERATED (For the period).
- NUMBER OF PACKETS GENERATED (For the period).
- AVERAGE PACKETS PER MESSAGE (For the period).
- NUMBER OF PACKETS COMPLETED TRIP (During the period).
- NUMBER OF PACKETS THAT LOOPED (During the period).
- TOTAL NUMBER OF PACKETS STILL IN NETWORK (At the end of the period).

c) A list of statistics of the packets that arrived to their destination during the period which includes the following information:

- AVG. LENGTH OF TRIP
  Average of "trip length" values of all packets that arrived to their final destination during the period, in "hops".
- AVG. TOTAL TRIP TIME
  Average of "trip time" values of all packets that arrived to their final destination during the period, in seconds.
- AVG. TIME SPENT IN QUEUE
  Average of "mean time spent in queue" values of all packets that arrived to their final destination during the period, in seconds.
- AVG. TIME PER HOP
  Average of "mean time per hop" values of all packets that arrived to their final destination during the period, in seconds.

3. PERIOD DATA

The routine 'COLLECT.PERIOD.DATA' collects status and statistical information during a period of the simulation (we will call it "Period Data"), and outputs it to 'UNIT 9', for its further graph and/or analysis. As in the case of Period Report, information of each period is independent from the rest. Data collected and output by this routine corresponds to the following information:

- Time of execution of the routine (or end of the period), in seconds.
- TOTAL NUMBER OF PACKETS STILL IN NETWORK (Same as Period Report).
- AVERAGE LINK UTILIZATION (Same as Period Report).
- AVERAGE QUEUE SIZE (Same as Period Report).
- MAXIMUM QUEUE SIZE (Same as Period Report).

- AVG. LENGTH OF TRIP (Same as Period Report).
- AVG. TOTAL TRIP TIME (Same as Period Report).
- AVG. TIME SPENT IN QUEUE (Same as Period Report).
- NUMBER OF PACKETS THAT LOOPED (Same as Period Report).

4. NETWORK REPORT

The event 'NET.REPORT', when executed, collects status and statistical information about the simulation and prints a report (we will call it "Network Report"), that resembles the Period Report. The main difference between this two reports is the time frame used to compute the data used by them. While for Period Report variables and totals used are reset at the end of each period, for Network Report they are never reset. In consecuence, each Network Report contains information for the "period" from time zero (start of simulation), up to its execution time.

The Network Report is divided in three sections, which contain the following information:

a) A list of all the links in the network (including their origin and destination nodes for ease of identification), indicating their utilization up to now; the mean, maximum value, and standard deviation of their "queue size" up to now; and the present size of their queue and propagation queue.

b) A list of the status and statistics of the network, which includes the following information:

   - AVERAGE LINK UTILIZATION

     Average calculated using the "link utilization" values of all links, up to now.

   - MAXIMUM LINK UTILIZATION

     Largest value observed, of the "Link Utilization" for all links, up to now.

   - AVERAGE QUEUE SIZE

Average calculated using the "mean queue size" of all links (in packets), up to now.

- MAXIMUM QUEUE SIZE

  The largest value observed, of the "queue size" of all links (in packets), up to now.

- STD.DEV. OF QUEUE SIZE

  The average value of the "standard deviation" of the "queue size" of all links, up to now.

- NUMBER OF MESSAGES GENERATED (Up to now).
- NUMBER OF PACKETS GENERATED (Up to now).
- AVERAGE PACKETS PER MESSAGE (Up to now).
- NUMBER OF PACKETS COMPLETED TRIP (Up to now).
- NUMBER OF PACKETS THAT LOOPED (Up to now).
- TOTAL NUMBER OF PACKETS STILL IN NETWORK (Up to now).
- AVERAGE PACKETS IN THE NETWORK (Up to now).

c) A list of statistics of the packets that arrived to their destination up to now, which includes the following information:

- AVG. LENGTH OF TRIP

  Average of "trip length" values of all packets that arrived to their final destination up to now, in "hops".

- AVG. TOTAL TRIP TIME

  Average of "trip time" values of all packets that arrived to their final destination up to now, in seconds.

- AVG. TIME SPENT IN QUEUE

  Average of "mean time spent in queue" values of all packets that arrived to their final destination up to now, in seconds.

- AVG. TIME PER HOP

  Average of "mean time per hop" values of all packets that arrived to their final destination up to now, in seconds.

5. NETWORK DATA

As in the case of the Network Report, the event 'DATA.COLLECTION' when executed, collects status and statistical information of the simulation (we will call it "Network Data"), for the "period" from time zero (start of simulation), up to its execution time. It then outputs this data to 'UNIT 8', for its further graphing and/or analysis. Data collected and output by this routine corresponds to the following information:

- Time of execution of the event, in seconds.
- TOTAL NUMBER OF PACKETS STILL IN NETWORK (Same as Network Report).
- AVERAGE LINK UTILIZATION (Same as Network Report).
- AVERAGE QUEUE SIZE (Same as Network Report).
- MAXIMUM QUEUE SIZE (Same as Network Report).
- AVG. LENGTH OF TRIP (Same as Network Report).
- AVG. TOTAL TRIP TIME (Same as Network Report).
- AVG. TIME SPENT IN QUEUE (Same as Network Report).
- NUMBER OF PACKETS THAT LOOPED (Same as Network Report).
- MAXIMUM LINK UTILIZATION (Same as Network Report).
- AVERAGE PACKETS IN THE NETWORK (Same as Network Report).

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The simulation was run at different traffic loads, which allowed us to determine the maximum load that the routing-fraction protocol could handle with this network configuration before saturating (we will call it saturation load). As shown in Figure D.1 of Appendix D, for the uniformly distributed traffic case with all link capacities set to 20,000 bits-per-second, saturation load was found to be 440 packets-per-second into the network. This load was then generated using different degrees of traffic granularity, i.e., different combinations of messages-per-second and packets-per-message (both for the network).

As shown in Table I, for the different degrees of traffic granularity tested (for the saturation load maintained at the same level) the running average of packets in the network, total time delay of packets that completed trip, average time spent in queue of packets that completed trip, and average queue size, where found to increase proportionally with the number of packets-per-message.

The maximum link utilization for the network was observed to decrease by a very small amount as we increased the number of packets-per-message. This reduction in link utilization is caused by the time factor that is included in its calculation (link utilization represents the proportion of time a link is involved in transmision of packets or "busy"), and the lower average number of messages received by the nodes for the same period. As we increase the number of packets-per-message while keeping the same saturation

TABLE I

GRANULARITY COMPARISON

	ROUTING FRACTIONS		
	440 msg/sec 1 pkt/msg	220 msg/sec 2 pkt/msg	110 msg/sec 4 pkt/msg
RUNNING AVERAGE PKTS IN NETWORK (in pkts)	116.17	252.04	468.06
TOTAL TIME DELAY PKTS COMPLETING TRIP (in secs)	0.260	0.562	1.056
AVERAGE TIME IN QUEUE PKTS COMPLETING TRIP (in secs)	0.164	0.466	0.962
AVERAGE QUEUE SIZE (in pkts)	1.22	3.48	7.09

level, we reduce the number of messages-per-second into the network. These changes will increase the message interarrival-time which is exponentially distributed with mean: (1 / messages-per-second). In addition for the same period, fewer nodes will receive new messages. The final effect is that on the average, there will be an increase in the time queues will be empty waiting for a new packet to arrive, and maximum link utilization will be slightly lower.

The running average number of packets in the network at the saturation level, for the different degrees of granularity, was found to be aproximatelly equal to the theoretical value given by Little's formula [Ref. 29], which states that:

$$E(N) = E(T) \times L$$

where:

E(N): Average value of packets in the network.

E(T): Average value of time delay.

L : Average packets per second entering the network.

This results confirm that the protocol behaves correctly for different traffic situations and conforms with the theoretical values expected.

A series of tests where performed using unbalanced traffic generation patterns, to investigate protocol performance under these conditions. The results of course showed that for unbalanced traffic situations performance degraded, due to the use of routing fractions which where calculated for a different (uniformly distributed) traffic load.

For the static routing case, then, an estimation of the traffic distribution is of primary importance if routing fractions are to be used, since the procedure used for their calculation will render a set of routing fractions which are optimal only for that traffic distribution. This result allows us to anticipate that for the dynamic case (where recalculation of the routing fractions to be used by the protocol will be done when variations in traffic characteristics or topology in the network warrant it), the protocol should render as good results as for the static case, since this recalculation procedure will maintain the routing fractions suited for existing conditions in the network. The main consideration will then be to determine when this routing fraction recalculation should be done.

A minimum number of hops (minimum-hops), single-path, static routing protocol was also calculated, as indicated in Appendix C. Simulation runs where performed for the minimum-hops and routing-fraction protocols under the same conditions to determine their relative performance. Results showed that the maximum load that the minimum-hops protocol could handle before saturation was 335 packets-per-second

into the network (see Figure D.2 of Appendix D). This load is 23% lower than the one found for the routing-fraction protocol.

As shown in Table II and Figures D.3 to D.6 of Appendix D, performance of the routing-fraction protocol for the 335 packets-per-second load was found to be significantly better than the performance of the minimum-hops protocol with the same traffic load.

TABLE II

PERFORMANCE COMPARISON

	ROUTING FRACTION 335 msg/sec 1 pkt/msg	MINIMUM HOPS 335 msg/sec 1 pkt/msg
RUNNING AVERAGE PKTS IN NETWORK (in pkts)	50.31	102.34
TOTAL TIME DELAY PKTS COMPLETING TRIP (in secs)	0.148	0.301
AVERAGE TIME IN QUEJE PKTS COMPLETING TRIP (in secs)	0.052	0.211
AVERAGE QUEUE SIZE (in pkts)	0.297	1.198

From the different status and statistical information gathered by the events and routines of the simulation program, the "Running Average of Packets in the Network", "Average Total Trip Time", "Average Time in Queue", and "Average Queue Size", where found to give the best indication of the way the protocol handled the traffic

during the simulation, and therefore were prefered as performance measures and for comparison between protocols. In a similar way "Maximum Link Utilization" and "Total Number of Packets Still in the Network", gave us a better indication of the maximum traffic loads the routing protocol could handle.In addition "Number of Packets that Completed Trip" when compared with "Number of Packets Generated" indicated the protocol's throughput performance.

Throughout our research the use of the graphing programs of Appendix F and Appendix G proved to be extremely useful since they provided an excellent means of visualizing and comparing large amounts of information. A sample output of some of the plots these programs can produce is shown in Appendix D.

A comparison was done between the virtual circuit service and datagram service configurations. For this comparison we used the successive saturation set of routing fractions, uniformly distributed traffic and link capacities set to 20,000 bits-per-second. Results showed that there is no significant difference between the two. This similarity is explained by the fact that the routing fractions used remain fixed throughout the simulation run and thus no difference is introduced by selecting the complete route of the message at the time it is generated.

All the tests mentioned previously where done using a set of routing fractions generated by the Successive Saturation method. As indicated in Chapter III Section E, another way of calculating routing fractions is the Maximum Slack method. In order to determine their relative performance, several simulation tests where performed using routing fractions calculated by the Successive Saturation method, and then repeated under the same conditions using routing fractions calculated by the Maximum Slack method.

Results showed that for a network with excess carrying capacity, i.e., traffic load much lower than the saturation load, the Maximum Slack set of routing fractions performed equally to the Successive Saturation set. For the case of traffic load near the saturation load, the Successive Saturation set performed better, and even gave a higher level of saturation load. This difference in performance is explained by the fact that when calculating the routing fractions using the Maximum Slack method, we maximize the sum of the slack saturation-ratio of all links, which tends to saturate more the links with higher capacity. In contrast, in the Successive Saturation method we tend to distribute saturation more uniformly among the links.

The maximum-slack routing fractions will therefore tend to use the excess capacity of the network, when there is some, and its performance then approaches that of the successive-saturation set. When there is no excess capacity in the network (near the saturation load), the routing fractions calculated by the successive-saturation method make better use of the available capacity, by spreading the traffic more uniformly over the network, and thus have a better performance.

Comparison was also made between the maximum slack set of routing fractions and the minimum-hops protocol. The saturation load of maximum-slack routing fractions was 420 packets-per-second, while for minimum-hops protocol it was 335 packets-per-second, which is 19% lower than the former. As shown in Table III, for the 335 packets-per-second load we found that the maximum slack set of routing fractions also rendered a much better performance than the minimum-hops protocol.

From these results we may conclude that the routing-fraction protocol is a much better alternative for static routing applications than the minimum-hops, single

TABLE III

PERFORMANCE COMPARISON

	MAXIMUM SLACK 335 msg/sec 1 pkt/msg	MINIMUM HOPS 335 msg/sec 1 pkt/msg
RUNNING AVERAGE PKTS IN NETWORK (in pkts)	49.77	102.34
TOTAL TIME DELAY PKTS COMPLETING TRIP (in secs)	0.146	0.301
AVERAGE TIME IN QUEUE PKTS COMPLETING TRIP (in secs)	0.052	0.211
AVERAGE QUEUE SIZE (in pkts)	0.297	1.198

path routing protocol. In addition, the successive saturation method should always be used when working with traffic loads that are close to the saturation load, and the maximum slack method should only be used when there is enough excess capacity in the network (compared to the saturation load), or when simplicity in the calculation is required.

B. RECOMMENDATIONS FOR FUTURE STUDY

The present study was done keeping in mind the future use of some of its contents for the investigation of the performance of routing fractions in a dynamic routing application. Additions have been made, when possible, of mechanisms that will facilitate this future work. Furthermore, provisions have been made in the simulation

program, to allow for the insertion of a subprogram to recalculate the values of the routing fractions.

For the analysis of the static case, status and statistical information was obtained mainly from events 'NET.REPORT' and 'DATA.COLLECTION', which use a time basis from the start of the simulation up to event execution time ("total-time"). In contrast data from routines 'PERIOD.REPORT' and 'COLLECT.PERIOD.DATA', which use a time basis from the start of period to the end of the same period ("period-time"), was used mostly for validation of total-time data. Nevertheless we have designed the latter with similar capabilities for gathering status and statistical information about the simulation as the former. This is because we expect the period-time data to become the prefered source of information for the analysis of the dynamic case, given the periodical nature of its process. We can also anticipate that the queue size and link utilization measurements will play an important rol in the decision of when to recalculate the values of the routing fractions, since their values directly reflect when a link is being saturated.

The built-in capability of the simulation program to configure the network to provide Virtual Circuit service should also become an important subject of investigation for the dynamic case. This aspect becomes more evident if we recognize that for this case routing fractions will change their values from time to time, and thus mixtures of virtual circuits created using different sets of routing fractions will exist simultaneously in the network, the interactions among which should be studied.

APPENDIX A

GENERATION OF GEOMETRICALLY DISTRIBUTED NUMBER OF PACKETS

The Geometric distribution is closely related to the Binomial distribution. The variable in the Geometric distribution is the number of trials preceding the first success, in a sequence of independant Binomial trials with probability of occurence equal to "p". Its probability density function (PDF) is:

$$f(x) = \begin{cases} p\,(1-p)^{x-1} & \text{for } x = 1, 2, 3,... \\ 0 & \text{otherwise} \end{cases}$$

The mean of the Gecmetric distribution is (1/p) and the variance (1-p)/p2. A Geometrically distributed random number can be generated directly from a standard uniformly-distributed number, by mapping it to the Geometric cummulative distribution function (CDF). The CDF of the Geometric distribution is:

$$F(x) = \begin{cases} 1 - (1-p)^{x} & \text{for } x = 1, 2, 3,... \\ 0 & \text{for } x < 1 \end{cases}$$

Let "y" be a number uniformly distributed between 0.0 and 1.0, then:

$$y = 1 - (1-p)^{x}$$

$$1 - y = (1-p)^{x}$$

$$\ln(1-y) = x \ln(1-p)$$

$$x = \ln(1-y) \,/\, \ln(1-p)$$

Since "y" is uniformly distributed from 0.0 to 1.0, then (1-y) will also be uniformly distributed from 0.0 to 1.0, therefore:

$$x = \ln(y) / \ln(1-p)$$

We must then round "x" to the closest integer that is higher or equal to it (since the Geometric is a discrete distribution), to get the value of the random number we are looking for.

To determine the number of packets for each message generated during the simulation run, we follow this procedure, and assign the value of the number obtained to the variable 'NUM.PKTS'. This variable is then used by the program, to control the number of packets generated for the message.

APPENDIX B

LINK NAME ASSIGNMENT

LINK NAME	NODE ORIG	NODE TERMINATION
1	1	2
2	1	3
3	1	4
4	1	5
5	2	1
6	2	3
7	2	6
8	2	9
9	3	1
10	3	2
11	3	4
12	3	6
13	3	7
14	4	1
15	4	3
16	4	5
17	4	7
18	4	8
19	5	1
20	5	4
21	5	8
22	5	12
23	6	2
24	6	3
25	6	7
26	6	9
27	6	10
28	7	3
29	7	4
30	7	6
31	7	8
32	7	10
33	7	11
34	8	4
35	8	5
36	8	7
37	8	11
38	8	12
39	9	2
40	9	6
41	9	10
42	9	13
43	10	6
44	10	7
45	10	9
46	10	11
47	10	13
48	11	7
49	11	8
50	11	10
51	11	12
52	11	13
53	12	5
54	12	8
55	12	11
56	12	13

57	13	9
58	13	10
59	13	11
60	13	12

APPENDIX C

MINIMUM NUMBER OF HOPS SINGLE PATH ROUTING TABLE

To calculate the values of a minimum number of hops (minimum-hops) single path routing table for the network of our study (see Figure 4.1 of Chapter IV), where all links have equal weight, we used the networks symmetry to simplify the procedure. We therefore divided the process into the following 4 steps:

1) Find the sink-tree (set of routes for packets from all nodes to the node selected) for one of the nodes located at the external vertices of the network. The resulting sink-tree shown in Figure C.1, can then be transformed into a sink-tree for the rest of the nodes located at external vertices of the network, by simply rotating it.

2) Repeat the procedure mentioned before but this time for nodes located at internal vertices of the network. The resulting sink-tree is shown in Figure C.2.

3) Find the sink-tree for the node located at the center of the network. The resulting tree is shown in Figure C.3.

4) Using the foregoing sink-trees construct the corresponding routing table. The resulting table is shown in Table IV . The left column indicates the sending node, the upper row indicates the destination node, and the contents of the table indicates the node via which traffic has to be sent.

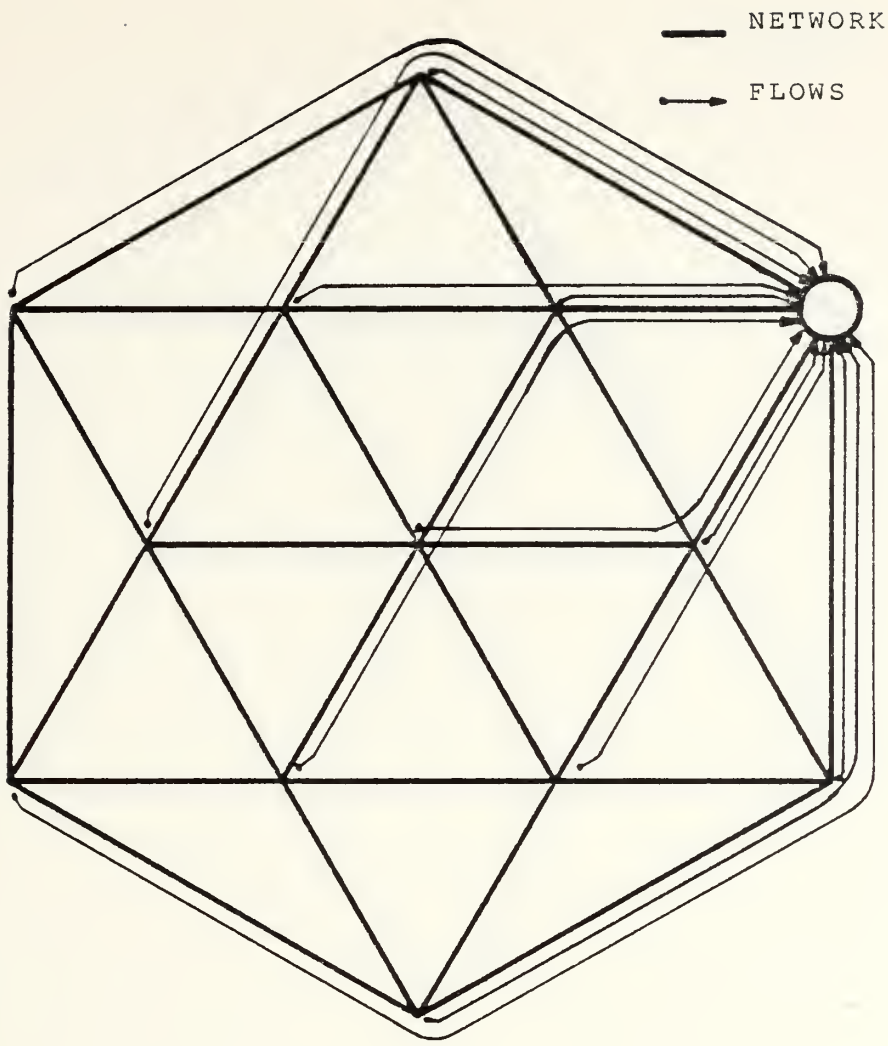

Figure C.1 SINK-TREE FOR EXTERNAL VERTICES.

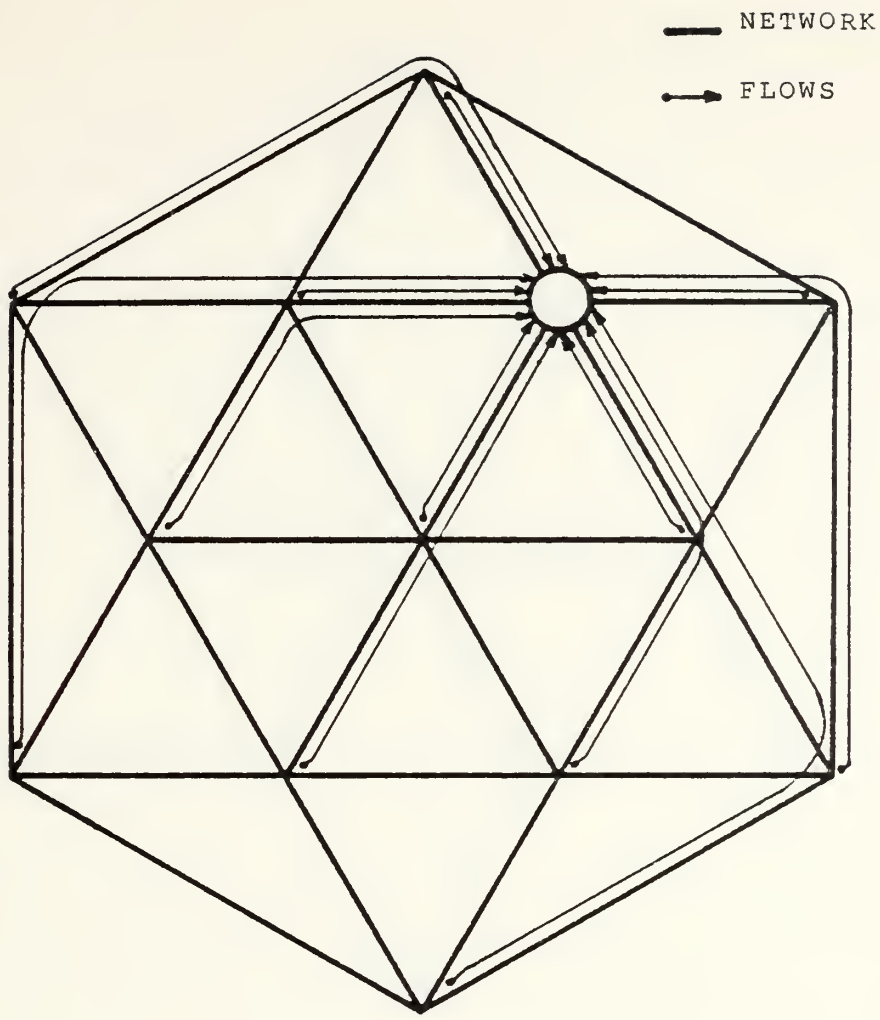

Figure C.2 SINK-TREE FOR INTERNAL VERTICES.

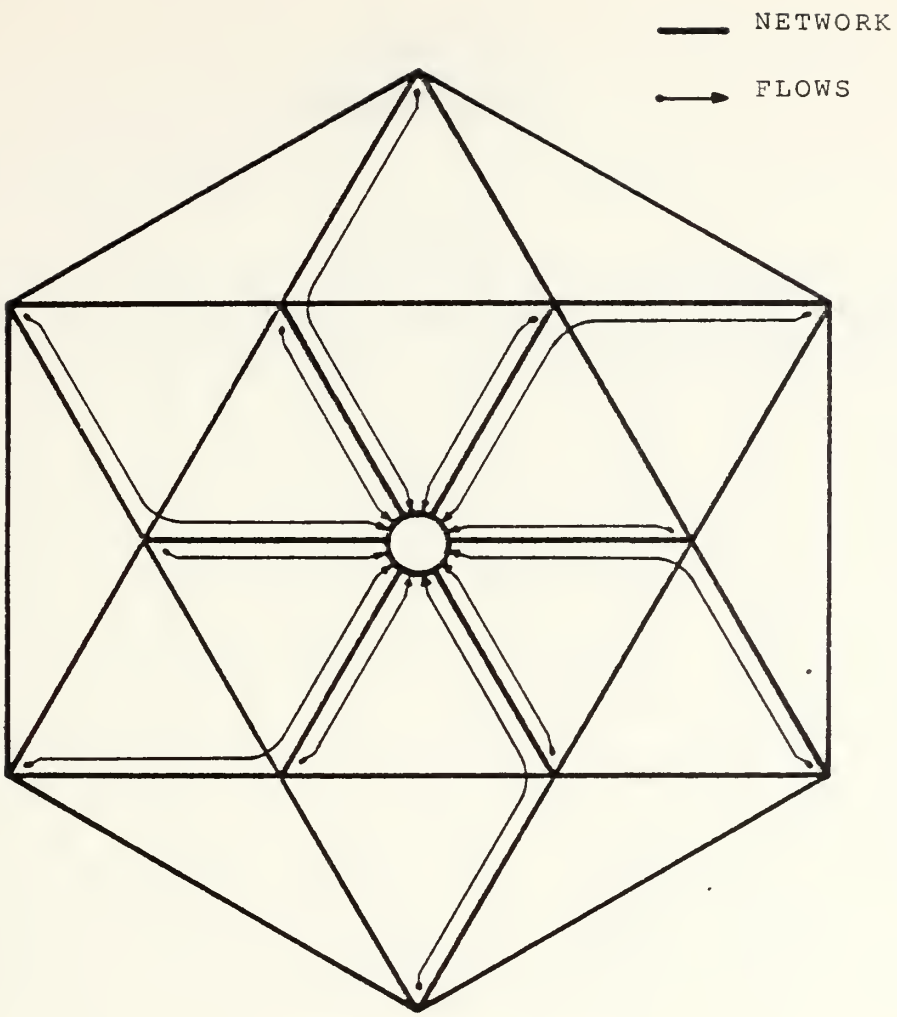

Figure C.3 SINK-TREE FOR CENTRAL NODE.

TABLE IV

ROUTING TABLE

FROM NODE	TO NODE												
	1	2	3	4	5	6	7	8	9	10	11	12	13
1	-	2	3	4	5	2	3	5	2	2	5	5	2
2	1	-	3	1	1	6	6	1	9	9	9	9	9
3	1	2	-	4	4	6	7	4	6	6	7	4	7
4	1	3	3	-	5	3	7	8	7	7	8	8	8
5	1	1	1	4	-	1	4	8	1	12	12	12	12
6	3	2	3	3	3	-	7	7	9	10	10	7	10
7	4	3	3	4	8	6	-	8	6	10	11	11	10
8	4	7	4	4	5	7	7	-	11	11	11	12	11
9	2	2	2	2	13	6	10	13	-	10	13	13	13
10	6	6	6	7	7	6	7	11	9	-	11	11	13
11	7	10	7	8	8	10	7	8	10	10	-	12	13
12	5	5	5	5	5	13	8	8	13	13	11	-	13
13	12	9	9	12	12	9	11	12	9	10	11	12	-

APPENDIX D

GRAPHIC REPRESENTATION OF DATA COLLECTED DURING SIMULATION

The following plots where obtained using the graphing programs of Appendixes E and F. Due to space constraints only the most important data is shown plotted.

Figure D.1 shows the number of packets still in the network for 440 packets-per-second (saturation load), and 460 packets-per-second (higher than saturation load) for the routing-fraction protocol.

Figure D.2 shows the number of packets still in the network for 335 packets-per-second (saturation load), and 350 packets-per-second (higher than saturation load) for the minimum number of hops (minimum-hops) protocol.

Figures D.3 to D.6 represent a graphic comparison of data collected during simulation of the routing-fraction and minimum-hops protocols, for the 335 packets-per-second load. The upper plot of each figure corresponds to data from the routing-fraction protocol, and the lower plot to data from the minimum-hops protocol.

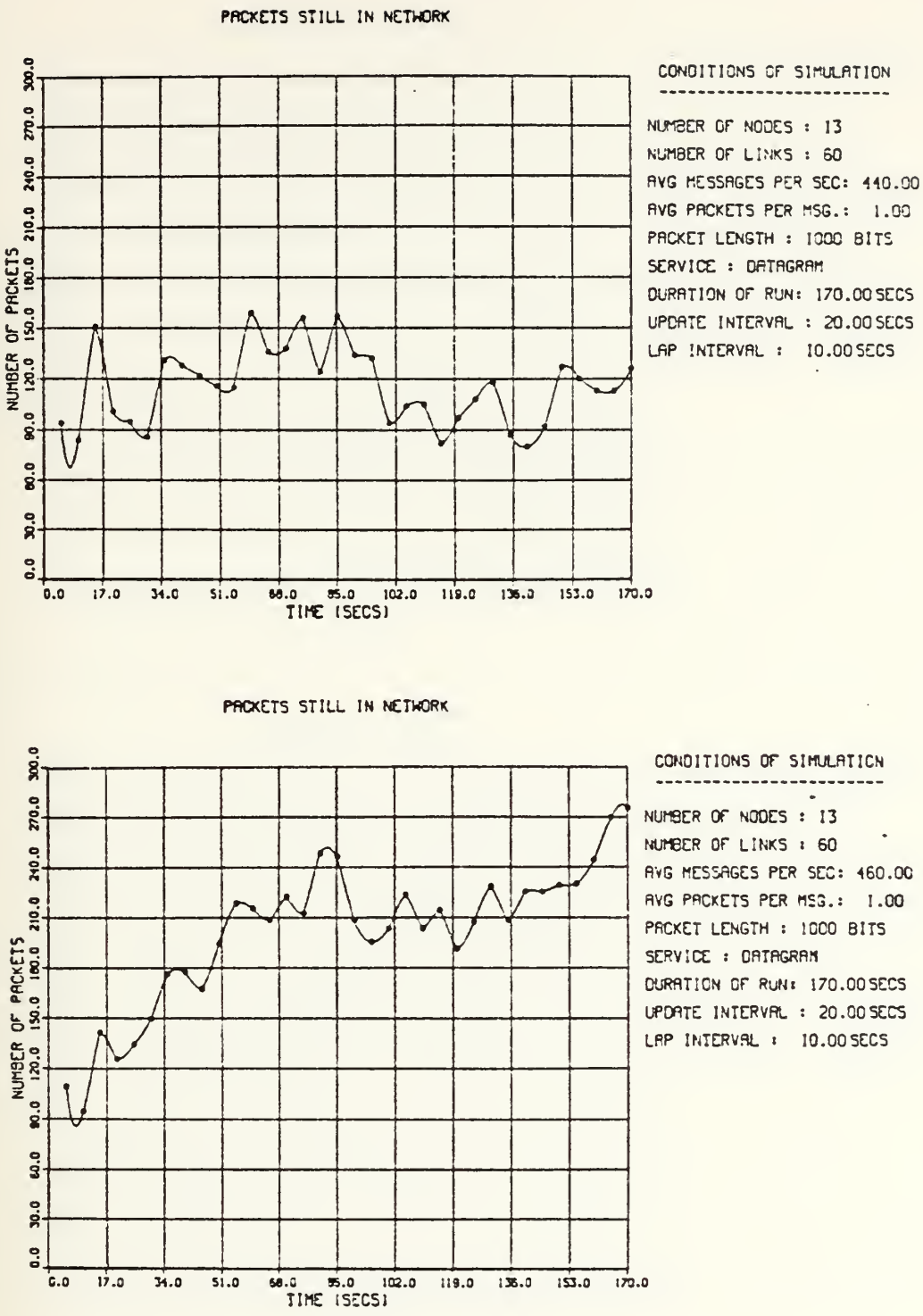

Figure D.1 PKTS STILL IN NETWORK FOR ROUTING-FRACTION PROTOCOL.

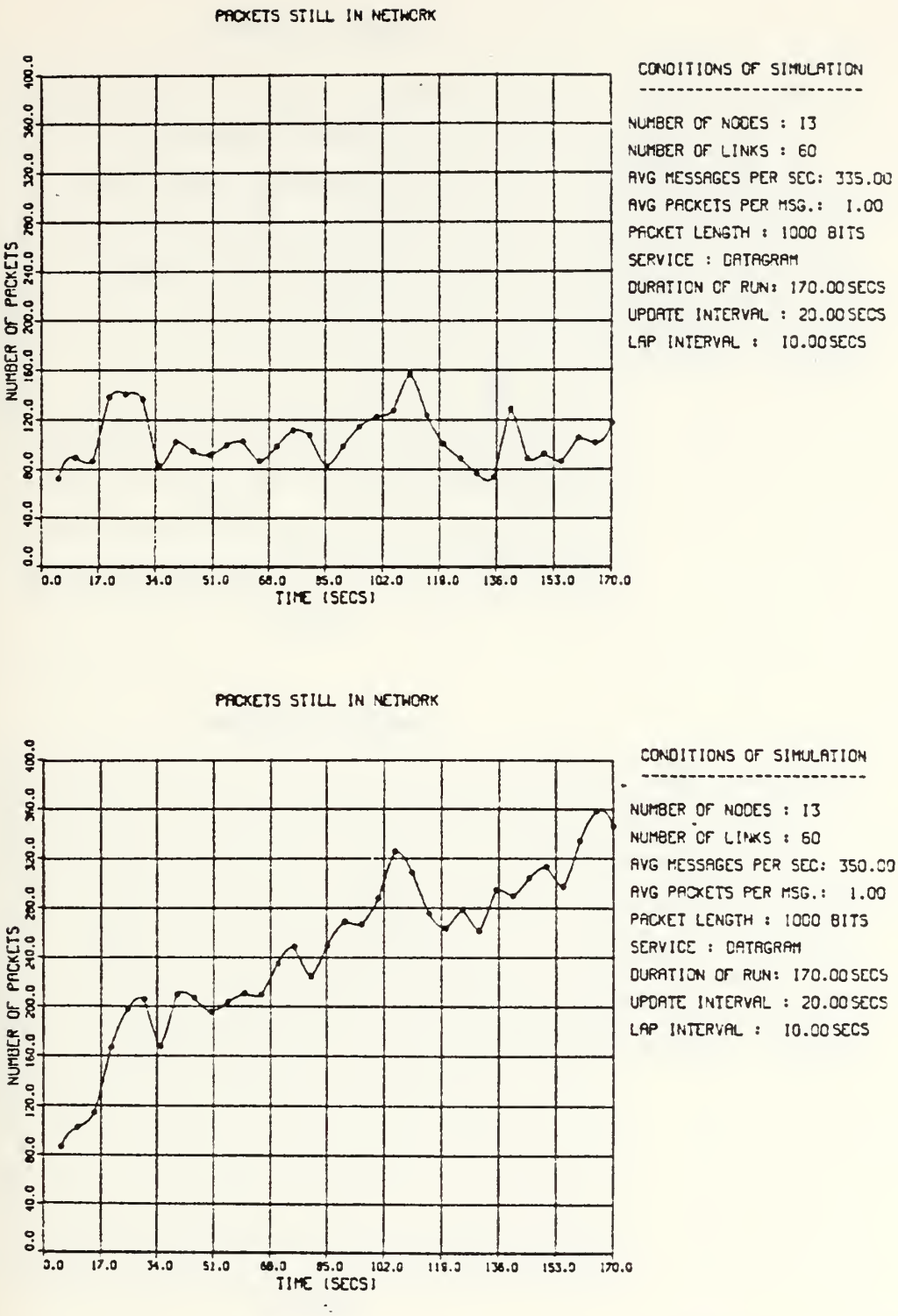

Figure D.2 PKTS. STILL IN NETWORK FOR MINIMUM-HOPS PROTOCOL.

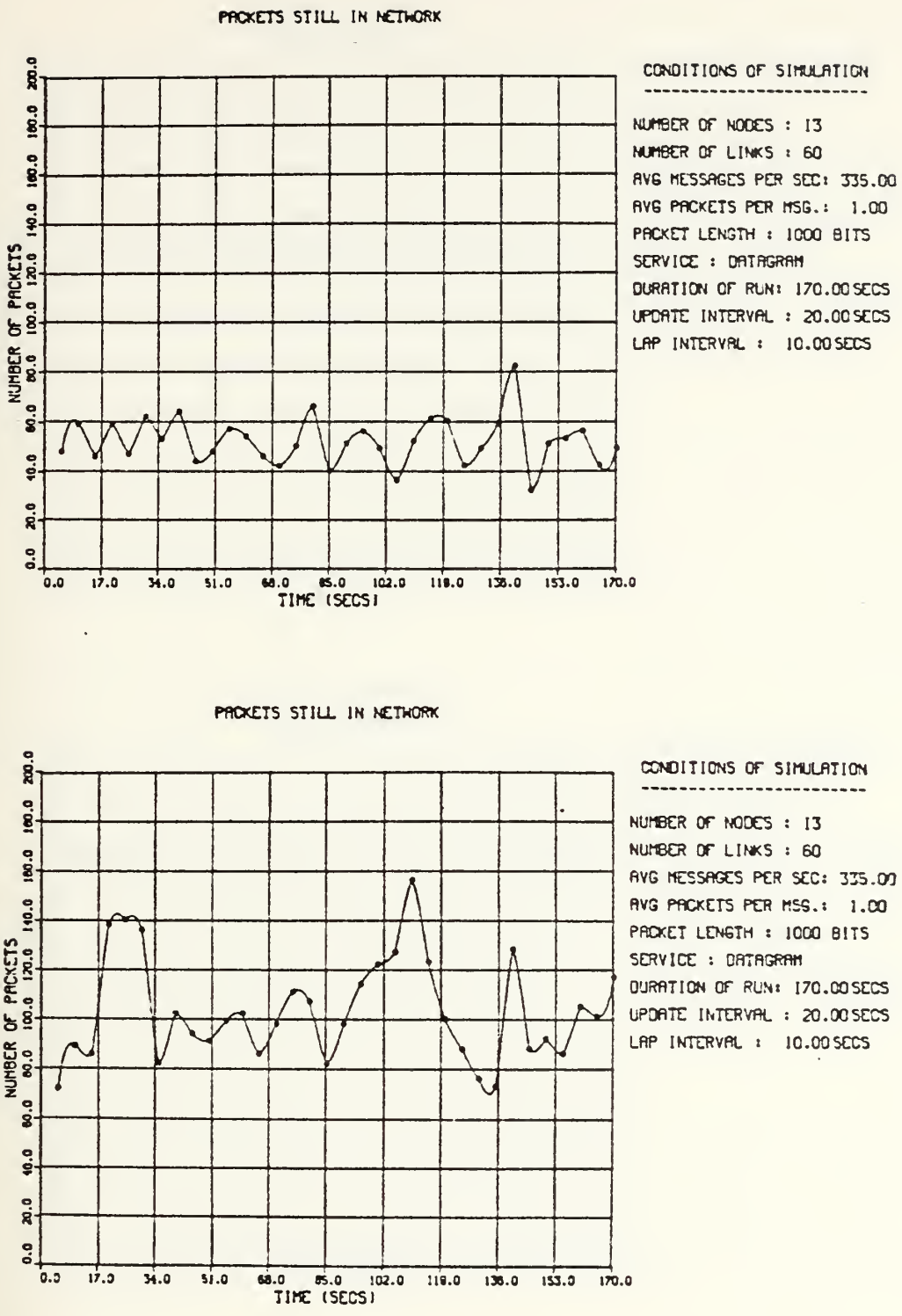

Figure D.3 COMPARISON OF PACKETS STILL IN THE NETWORK.

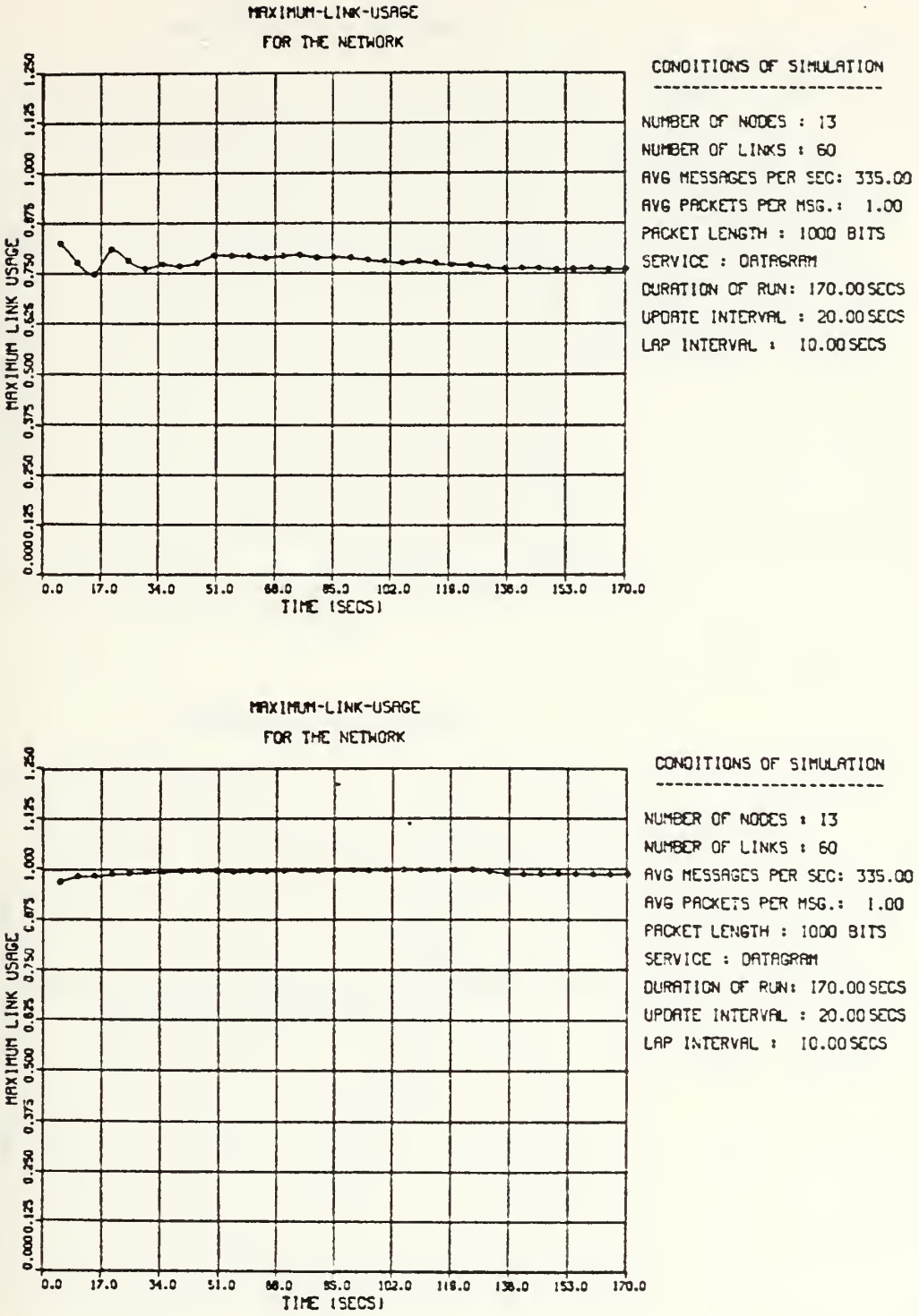
MAXIMUM-LINK-USAGE
FOR THE NETWORK
MAXIMUM LINK USAGE
0.000 0.125 0.250 0.375 0.500 0.625 0.750 0.875 1.000 1.125 1.250
0.0 17.0 34.0 51.0 68.0 85.0 102.0 118.0 138.0 153.0 170.0
TIME (SECS)
CONDITIONS OF SIMULATION
NUMBER OF NODES : 13
NUMBER OF LINKS : 60
AVG MESSAGES PER SEC: 335.00
AVG PACKETS PER MSG.: 1.00
PACKET LENGTH : 1000 BITS
SERVICE : DATAGRAM
DURATION OF RUN: 170.00 SECS
UPDATE INTERVAL : 20.00 SECS
LAP INTERVAL : 10.00 SECS
MAXIMUM-LINK-USAGE
FOR THE NETWORK
MAXIMUM LINK USAGE
0.000 0.125 0.250 0.375 0.500 0.625 0.750 0.875 1.000 1.125 1.250
0.0 17.0 34.0 51.0 68.0 85.0 102.0 118.0 138.0 153.0 170.0
TIME (SECS)
CONDITIONS OF SIMULATION
NUMBER OF NODES : 13
NUMBER OF LINKS : 60
AVG MESSAGES PER SEC: 335.00
AVG PACKETS PER MSG.: 1.00
PACKET LENGTH : 1000 BITS
SERVICE : DATAGRAM
DURATION OF RUN: 170.00 SECS
UPDATE INTERVAL : 20.00 SECS
LAP INTERVAL : 10.00 SECS

Figure D.4 COMPARISON OF MAXIMUM LINK USAGE FOR THE NETWORK.

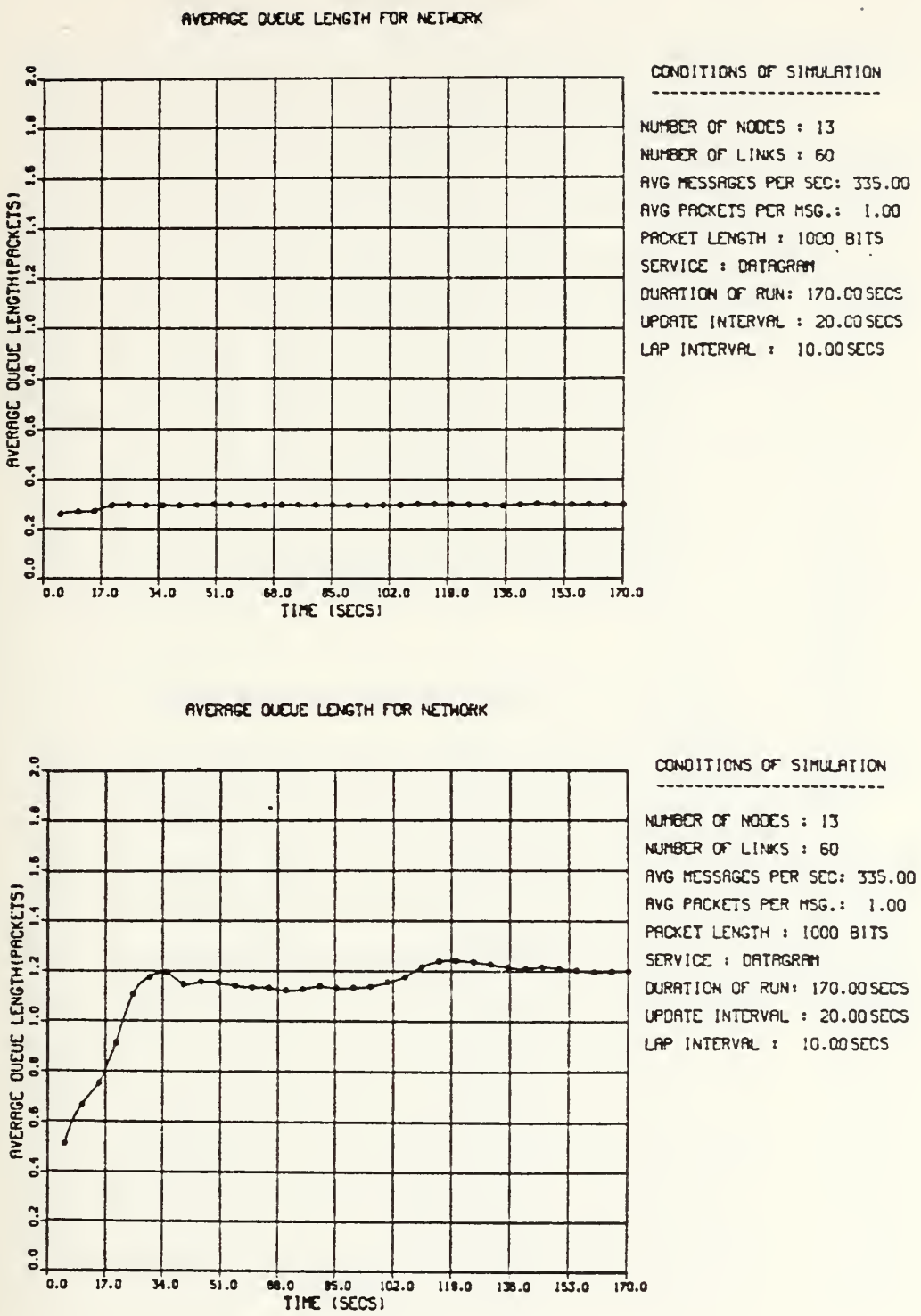

Figure D.5 COMPARISON OF AVERAGE QUEUE LENGTH FOR THE NETWORK.

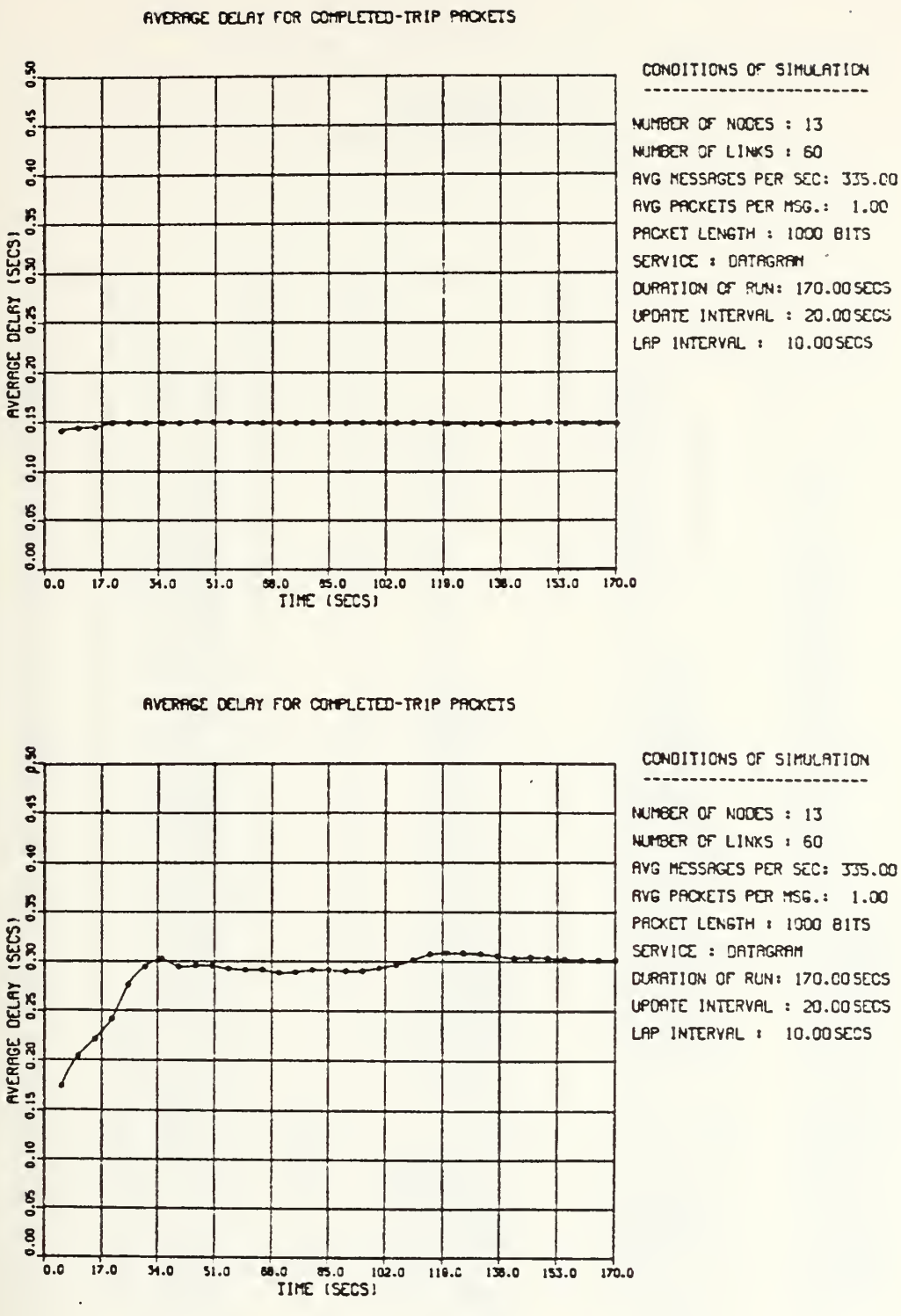

Figure D.6 COMPARISON OF AVG. DELAY FOR COMPLETING TRIP PKTS.

```
;; ##################### SIMULATION PROGRAM ##########################
;;
;; THE PURPOSE OF THIS PROGRAM, IS TO SIMULATE A PACKET SWITCHING
;; COMMUNICATIONS NETWORK THAT EMPLOYS ROUTING-FRACTIONS IN ITS
;; ROUTING PROTOCOL.
;;
;; ***************************************************************
PREAMBLE LAST COLUMN IS 70
;;
;; THE PREAMBLE CONTAINS MODE AND DIMENSION DEFINITIONS OF GLOBAL
;; VARIABLES, DECLARATION OF EVENTS AND EVENT PRIORITIES,
;; AND DEFINITION OF STATISTICAL VARIABLES AND PROGRAM FUNCTIONS
;;
;; -----------------------------------------------------------------
;;
GENERATE LIST ROUTINES
;;
;; -----------------------------------------------------------------
;;
PERMANENT ENTITIES
    EVERY NODE OWNS A LINK.SET
    EVERY LINK OWNS A QUEUE AND A PROP.QUEUE, HAS A LNK.NAME,
        A LNK.ORIG, A LNK.TERM, A LNK.STATUS
        AND BELONGS TO A LINK.SET
    DEFINE QUEUE AS A FIFO SET
    DEFINE PROP.QUEUE AS A FIFO SET
;;
;; -----------------------------------------------------------------
;;
TEMPORARY ENTITIES
    EVERY PACKET OWNS A TRIP.RECORD, HAS A PKT.ID, A PKT.BIRTH,
        A PKT.ORIG, A PKT.DEST, A HOP.COUNT, A LIFETIME, A QSTAT,
        MAY BELONG TO A QUEUE AND A PROP.QUEUE
    EVERY RECORD HAS A PAST.NODE AND BELONGS TO A TRIP.RECORD
    DEFINE TRIP.RECORD AS A FIFO SET
;;
;; -----------------------------------------------------------------
;;
EVENT NOTICES INCLUDE NEW.MSG, NET.REPORT, DATA.COLLECTION,
        NEW.ROUT.FRAC, LAP.TOTALS.RESET, DESTRUCTION
    EVERY START.XMT HAS A LINK
    EVERY ARRIVAL HAS AN ARR.LNK
    EVERY END.XMT HAS A XMT.LINK
;;
;; -----------------------------------------------------------------
;;
PRIORITY ORDER IS ARRIVAL, START.XMT, END.XMT, DATA.COLLECTION,
    NET.REPORT, NEW.ROUT.FRAC, DESTRUCTION, NEW.MSG,
```

```
     LAP.TOTALS.RESET
;;
;; ------------------------------------------------------------
;;
ACCUMULATE TRANSIT.TIME AS THE SUM OF LIFETIME
ACCUMULATE QUE.PKT.TIME AS THE SUM OF QSTAT
ACCUMULATE UTILZ.PERIOD AS THE PERIODICAL MEAN, LAP.UTILZ AS
     THE LAP MEAN AND LK.MEAN AS THE GRAND MEAN OF LNK.STATUS
ACCUMULATE QU.MAX AS THE GRAND MAXIMUM, QU.MEAN AS THE GRAND MEAN,
     QU.DEV AS THE GRAND STD.DEV, PER.MAXQ AS THE PERIODICAL MAXIMUM,
     PER.QMEAN AS THE PERIODICAL MEAN  AND
     PER.DEVQ AS THE PERIODICAL STD.DEV  OF N.QUEUE
ACCUMULATE AVG.NET.PKTS AS THE MEAN OF NET.PKTS
;;
;; ------------------------------------------------------------
;;
DEFINE SELECT.NODES AS AN INTEGER FUNCTION
DEFINE PICK.BEST.LINK AS AN INTEGER FUNCTION
;;
;; ------------------------------------------------------------
;;
DEFINE CONNECTED AS A 2-DIMENSIONAL, INTEGER ARRAY
DEFINE LNK.CAPACITY AS A 1-DIMENSIONAL, INTEGER ARRAY
DEFINE QSIZE AS A 1-DIMENSIONAL, INTEGER ARRAY
DEFINE BE.LI AS A 1-DIMENSIONAL, INTEGER ARRAY
DEFINE VIR.CI AS A 1-DIMENSIONAL, INTEGER ARRAY
;;
;; ------------------------------------------------------------
;;
DEFINE PR.DEST AS A 2-DIMENSIONAL, REAL ARRAY
DEFINE PR.PAIR AS A 2-DIMENSIONAL, REAL ARRAY
DEFINE ROUT.FRAC AS A 2-DIMENSIONAL, REAL ARRAY
DEFINE UTZ.PERIOD AS A 1-DIMENSIONAL, REAL ARRAY
DEFINE UTZ.LAP AS A 1-DIMENSIONAL, REAL ARRAY
DEFINE PR.ORIG AS A 1-DIMENSIONAL, REAL ARRAY
;;
;; ------------------------------------------------------------
;;
DEFINE LNK.NAME, LNK.ORIG, LNK.TERM, LNK.STATUS, PKT.ID, PKT.ORIG,
     PKT.DEST, HOP.COUNT, LIFETIME, QSTAT, PAST.NODE, LENGTH.PKT,
     COMP.TRIP.PKT, TOTAL.LOOP.PKTS, MSG.TOTAL, PKT.TOTAL, HOP.TOTAL,
     PAIRS, PRNT, LI.NK, XMT.LINK, ARR.LNK, MO.DE, MSG.PERIOD,
     PKT.PERIOD, HOP.PERIOD, PKTS.ARR.PERIOD, LOOP.PKTS.PERIOD,
     NET.PKTS   AS INTEGER VARIABLES
;;
;; ------------------------------------------------------------
;;
DEFINE PKT.BIRTH, AVG.PKTS.MSG, TIME.LIMIT, UP.TIME, COLL.INT,
```

```
    REPT.TIME, TRIP.TOTAL, QU.TOTAL, AVG.MPS.NET, LAP.TIME,
    LAST.PERIOD, TRIP.PERIOD, QU.PERIOD
    AS REAL VARIABLES
''
'' ------------------------------------------------------------
''
END  '' OF PREAMBLE
''
'' ************************************************************
MAIN
''
'' THE MAIN PROGRAM IS THE DRIVER OF THE SIMULATION, IT CALLS AND
'' SCHEDULES THE INITIALIZING ROUTINES AND EVENTS, STARTS AND ENDS
'' THE SIMULATION.
''
'' ------------------------------------------------------------
''
PERFORM NETWORK.CONSTRUCTION
PERFORM INITIALIZATION
PERFORM PRINT.INITIAL.CONDITIONS
SCHEDULE A NEW.MSG IN EXPONENTIAL.F((1 / AVG.MPS.NET),8) UNITS
IF UP.TIME GT 0,
  SCHEDULE A NEW.ROUT.FRAC IN UP.TIME UNITS
ALWAYS
IF COLL.INT GT 0,
  SCHEDULE A DATA.COLLECTION IN COLL.INT UNITS
ALWAYS
IF REPT.TIME GT 0,
  SCHEDULE A NET.REPORT IN REPT.TIME UNITS
ALWAYS
IF LAP.TIME GT 0,
  SCHEDULE A LAP.TOTALS.RESET IN LAP.TIME UNITS
ALWAYS
SCHEDULE A DESTRUCTION IN TIME.LIMIT UNITS
START SIMULATION
STOP
END  '' OF MAIN
''
'' ************************************************************
ROUTINE FOR NETWORK.CONSTRUCTION
''
'' THIS ROUTINE READS THE NUMBER OF NODES AND LINKS AND CONSTRUCTS
'' THE NETWORK BASED ON THOSE PARAMETERS.
''
'' ------------------------------------------------------------
DEFINE X, Y, Z, K, I, J
    AS INTEGER VARIABLES
'' ------------------------------------------------------------
```

```
''
READ N.NODE
     CREATE EVERY NODE
READ N.LINK
     CREATE EVERY LINK
RESERVE CONNECTED(*,*) AS N.LINK BY N.NODE
LET X = 0
LET Y = 0
LET Z = 0
LET K = 0
FOR I=1 TO N.LINK, DO
     FOR J=1 TO N.NODE, DO
        READ CONNECTED(I,J)
        IF CONNECTED(I,J) EQ 1,
           LET X = J
        ALWAYS
        IF CONNECTED(I,J) EQ (-1),
           LET Y = J
        ALWAYS
     LOOP
     LET Z = Z + 1
     LET K = K + 1
     LET LNK.NAME(K) = Z
     LET LNK.ORIG(K) = X
     LET LNK.TERM(K) = Y
     FILE K IN LINK.SET(X)
LOOP
RETURN
END  '' OF NETWORK.CONSTRUCTION
''
'' ***************************************************************
ROUTINE FOR INITIALIZATION
''
'' THIS ROUTINE INPUTS VALUES OF INITIAL CONDITIONS CHOSEN BY USER
'' AND INITIALIZES GLOBAL VARIABLES FOR SIMULATION RUN
''
'' -------------------------------------------------------------
DEFINE Z, K, I, J, LNK, THE.NAME, THE.ORIG, THE.TERM
     AS INTEGER VARIABLES
'' -------------------------------------------------------------
DEFINE G, H, C
     AS REAL VARIABLES
'' -------------------------------------------------------------
''
'INIT.PR.PAIR'
''
RESERVE PR.ORIG(*) AS N.NODE
RESERVE PR.DEST(*,*) AS N.NODE BY N.NODE
```

```
''
LET PAIRS = 0
FOR K = 1 TO N.NODE, DO
    READ PR.ORIG (K)
LOOP
FOR I = 1 TO N.NODE, DO
    FOR J = 1 TO N.NODE, DO
        READ PR.DEST (I,J)
        LET G = PR.ORIG (I) * PR.DEST (I,J)
        IF (G GT 0) AND (I NE J),
            LET PAIRS = PAIRS + 1
        ALWAYS
    LOOP
LOOP
''
RESERVE PR.PAIR(*,*) AS PAIRS BY 3
''
LET Z = 0
LET Q = 0.0
FOR I = 1 TO N.NODE, DO
    FOR J = 1 TO N.NODE, DO
        LET H = PR.ORIG(I) * PR.DEST(I,J)
        IF (H GT 0) AND (I NE J),
            LET Z = Z + 1
            LET Q = Q + H
            LET PR.PAIR(Z,1) = Q
            LET PR.PAIR(Z,2) = REAL.F(I)
            LET PR.PAIR(Z,3) = REAL.F(J)
        ALWAYS
    LOOP
LOOP
''
'INIT.ROUT.FRAC'
RESERVE ROUT.FRAC(*,*) AS N.LINK BY N.NODE
FOR I=1 TO N.LINK, DO
    FOR J=1 TO N.NODE, DO
        READ ROUT.FRAC(I,J)
    LOOP
LOOP
''
'INIT.LNK.CAPACITY'
RESERVE LNK.CAPACITY(*) AS N.LINK
FOR I=1 TO N.LINK, DO
    READ LNK.CAPACITY(I)
LOOP
''
'INPUT.VARIOUS'
''
```

```
READ TIME.LIMIT
READ REPT.TIME
READ UP.TIME
READ COLL.INT
READ LAP.TIME
READ LENGTH.PKT
READ AVG.MPS.NET
READ AVG.PKTS.MSG
READ PRNT
READ MO.DE
''
'INIT.VARIOUS'
''
LET MSG.TOTAL = 0
LET MSG.PERIOD = 0
LET PKT.TOTAL = 0
LET PKT.PERIOD = 0
LET COMP.TRIP.PKT = 0
LET PKTS.ARR.PERIOD = 0
LET TOTAL.LOOP.PKTS = 0
LET LOOP.PKTS.PERIOD = 0
LET HOP.TOTAL = 0
LET HOP.PERIOD = 0
LET NET.PKTS = 0
LET TRIP.TOTAL = 0.0
LET TRIP.PERIOD = 0.0
LET QU.TOTAL = 0.0
LET QU.PERIOD = 0.0
''
RETURN
END  '' OF INITIALIZATION
''
'' ************************************************************
ROUTINE TO PRINT.INIT.CONDITIONS
''
'' THIS ROUTINE PRINTS INITIAL SETUP OF NETWORK AND INITIAL
'' CONDITIONS FOR SIMULATION RUN
''
'' ------------------------------------------------------------
DEFINE I, J, ORIG, DEST
    AS INTEGER VARIABLES
'' ------------------------------------------------------------
''
SKIP 2 OUTPUT LINES
PRINT 1 LINE AS FOLLOWS
##########SIMULATION INITIAL CONDITIONS##########
PRINT 12 LINES WITH N.NODE, N.LINK, TIME.LIMIT, REPT.TIME, UP.TIME,
COLL.INT, LAP.TIME, LENGTH.PKT, AVG.MPS.NET, AVG.PKTS.MSG, PRNT,
```

```
MO.DE  AS FOLLOWS
NUMBER OF NODES:***NODES
NUMBER OF LINKS:***LINKS
DURATION OF SIMULATION:*******.****SECS
REPORT GENERATION INTERVAL:*****.****SECS
ROUTING UPDATE INTERVAL:*****.****SECS
DATA COLLECTION INTERVAL:*****.****SECS
LAP INTERVAL:*****.****SECS
PACKET LENGTH:******BITS
AVG. MESSAGES PER SECOND FOR NETWORK:****.*** MESSAGES
AVG. PACKETS PER MESSAGE FOR NETWORK:****.*** PACKETS
PRINT CONDITICN:**
MODE SELECTED:**
SKIP 1 OUTPUT LINE
PRINT 4 LINES AS FOLLOWS
------------------------------ LINKS ------------------------------

              LINK        NODE          NODE          LINK
              NAME        ORIG      TERMINATION       CAPACITY
FOR EACH LINK, DO
    PRINT 1 LINE WITH LINK, LNK.ORIG(LINK), LNK.TERM(LINK),
       LNK.CAPACITY(LINK)  AS FOLLOWS
              ****        ***           ***           *******BPS
LOOP
SKIP 1 OUTPUT LINE
PRINT 4 LINES AS FOLLOWS
------------------------ ROUTING FRACTIONS ------------------------

               FROM         TO         VIA         ROUTING
               NODE        NODE       NODE        FRACTION
FOR EACH LINK, DO
    FOR J=1 TO N.NODE, DO
        IF ROUT.FRAC(LINK,J) NE 0,
           PRINT 1 LINE WITH LNK.ORIG(LINK), J, LNK.TERM(LINK),
              ROUT.FRAC(LINK,J)
              AS FOLLOWS
             ***         ***         ***          *.****
        ALWAYS
    LOOP
LOOP
SKIP 1 OUTPUT LINE
PRINT 4 LINES AS FOLLOWS
---------------- PROBABILITY OF NODE PAIR SELECTION ----------------

      NODE-PAIR     NODE    NODE     PROBABILITY      CUMMULATIVE
       NUMBER       ORIG    DEST     OF SELECTION     PROBABILITY
FOR I=1 TO PAIRS, DO
    LET ORIG = TRUNC.F(PR.PAIR(I,2) + 0.1)
```

```
    LET DEST = TRUNC.F(PR.PAIR(I,3) + 0.1)
    PRINT 1 LINE WITH I, ORIG, DEST,
       (PR.ORIG(ORIG) * PR.DEST(ORIG,DEST)), PR.PAIR(I,1) AS FOLLOWS
        *****      ***    ***        *.*****          *.*****
LOOP
SKIP 4 OUTPUT LINES
USE UNIT 8 FOR OUTPUT
   WRITE N.NODE, N.LINK, MO.DE, LENGTH.PKT, AVG.MPS.NET, TIME.LIMIT,
     UP.TIME, LAP.TIME, COLL.INT, AVG.PKTS.MSG
     AS /,B 1,2 I(4),I(2),I(5),6 D(8,3)
USE UNIT 9 FOR OUTPUT
   WRITE N.NODE, N.LINK, MO.DE, LENGTH.PKT, AVG.MPS.NET, TIME.LIMIT,
     UP.TIME, LAP.TIME, AVG.PKTS.MSG
     AS /,B 1,2 I(4),I(2),I(5),5 D(8,3)
USE UNIT 6 FOR OUTPUT
RETURN
END  '' OF PRINT.INITIAL.CONDITIONS
''
'' ************************************************************
ROUTINE TO SELECT.NODES
''
'' THIS ROUTINE SELECTS A NODE-PAIR NUMBER AND RETURNS IT AS 'LOW'
''
'' ------------------------------------------------------------
DEFINE LOW, MID, HIGH
    AS INTEGER VARIABLES
'' ------------------------------------------------------------
DEFINE SAMPLE
    AS REAL VARIABLE
'' ------------------------------------------------------------
''
LET SAMPLE = UNIFORM.F(0.0,1.0,1)
LET LOW = 1
LET HIGH = PAIRS
UNTIL LOW EQ HIGH, DO
    LET MID = TRUNC.F((LOW +HIGH + 0.1)/2)
    IF SAMPLE GT PR.PAIR(MID,1)
       LET LOW = MID + 1
    ELSE
       IF SAMPLE LT PR.PAIR(MID,1),
          LET HIGH = MID
       ELSE
          GO OUT
       ALWAYS
    ALWAYS
LOOP
'OUT'
IF PRNT GE 1,
```

```
PRINT 1 LINE WITH SAMPLE, LOW AS FOLLOWS
    SAMPLE(TO SEL NODES)= **.******   LOW= ***
ALWAYS
RETURN WITH LOW
END  '' OF SELECT.NODES
''
'' ************************************************************
ROUTINE TO PICK.BEST.LINK GIVEN NO.DE, DE.ST
''
'' THIS ROUTINE SELECTS BEST OUTGOING LINK GIVEN NODE WHERE PACKET
'' IS AND PACKET DESTINATION. NAME OF BEST LINK IS RETURNED AS
'' 'BST.PCK'
''
'' ------------------------------------------------------------
DEFINE THE.NAME, NO.DE, DE.ST, BST.PCK
    AS INTEGER VARIABLES
'' ------------------------------------------------------------
DEFINE Q, H
    AS REAL VARIABLES
'' ------------------------------------------------------------
''
'PICK'
LET Q = UNIFORM.F(0.0,1.0,2)
''
IF PRNT GE 1,
  PRINT 1 LINE WITH Q AS FOLLOWS
    Q(UNIF TO SEL LNK)=**.*****
ALWAYS
''
LET H = 0.0
FOR EACH LINK IN LINK.SET(NO.DE), DO
    LET THE.NAME = LNK.NAME(LINK)
    IF PRNT GE 1,
      PRINT 1 LINE WITH THE.NAME AS FOLLOWS
    THE.NAME=***
    ALWAYS
    IF ROUT.FRAC(THE.NAME,DE.ST) NE 0,
        LET H = H + ROUT.FRAC(THE.NAME,DE.ST)
        IF Q LE H,
           LET BST.PCK = THE.NAME
           GO OUT
        REGARDLESS
    ALWAYS
LOOP
'OUT'
IF LNK.CAPACITY(BST.PCK) LE 0,
   GO PICK
REGARDLESS
```

```
IF PRNT GE 1,
  PRINT 1 LINE WITH BST.PCK AS FOLLOWS
    BEST.PCK=***
ALWAYS
RETURN WITH BST.PCK
END  '' OF PICK.BEST.LINK
''
'' **************************************************************
ROUTINE TO SET.VIRTUAL.CIRCUIT GIVEN  START.NODE  AND  END.NODE
      YIELDING  Y  AND  BE.LI
''
'' THIS ROUTINE FINDS VIRTUAL CIRCUIT GIVEN MESSAGE ORIGIN AND
'' DESTINATION. THE NUMBER OF LINKS IN VIRTUAL CIRCUIT IS RETURNED
'' AS 'Y' AND NAME OF SUCCESIVE LINKS AS 'BE.LI(*)'
''
'' --------------------------------------------------------------
DEFINE START.NODE, END.NODE, X, Y, I
   AS INTEGER VARIABLES
'' --------------------------------------------------------------
''
RESERVE BE.LI(*) AS N.LINK
LET X = START.NODE
LET Y = 1
'AGAIN'
LET BE.LI(Y) = PICK.BEST.LINK(X,END.NODE)
FOR I=1 TO (Y - 1), DO
    IF LNK.TERM(BE.LI(Y)) EQ LNK.TERM(BE.LI(I)),
       GO AGAIN
    REGARDLESS
LOOP
IF LNK.TERM(BE.LI(Y)) EQ END.NODE,
   GO OUT
REGARDLESS
LET X = LNK.TERM(BE.LI(Y))
LET Y = Y + 1
GO AGAIN
'OUT'
RETURN
END  '' OF SET.VIRTUAL.CIRCUIT
''
'' **************************************************************
ROUTINE PERIOD.REPORT
''
'' THIS ROUTINE CALCULATES AND PRINTS STATUS AND STATISTICAL DATA
'' OF A PERIOD. STARTING TIME OF PERIOD IS TAKEN FROM 'LAST.PERIOD'
'' AND ENDING TIME FROM 'TIME.V'. ROUTINE 'RESTART.PERIOD.TOTALS'
'' REINITIALIZES TOTALS AND VARIABLES USED IN PERIOD SO STATS AND
'' STATISTICAL DATA OF EACH PERIOD IS INDEPENDENT OF THE REST.
```

```
''
'' ------------------------------------------------------------
DEFINE MX.QU
    AS INTEGER VARIABLE
'' ------------------------------------------------------------
DEFINE MN.LNK, MN.QUE, DEV.QUE
    AS REAL VARIABLES
'' ------------------------------------------------------------
''
LET MN.LNK = 0.0
LET MN.QUE = 0.0
LET DEV.QUE = 0.0
LET MX.QU = 0
SKIP 2 LINES
PRINT 4 LINES WITH LAST.PERIOD, TIME.V AS FOLLOWS
#### NETWORK REPORT FOR PERIOD FROM ****.***** TO ****.***** SECS ###

-----------LINK---------  ==========QUEUE SIZE==========  ---PROP---
NAME FROM TO  UTILIZATION     MEAN      MAX      DEV      NOW  QUEUE(NOW)
''
FOR EACH LINK, DO
    PRINT 1 LINE WITH LNK.NAME(LINK), LNK.ORIG(LINK),
        LNK.TERM(LINK), UTILZ.PERIOD(LINK), PER.QMEAN(LINK),
        PER.MAXQ(LINK), PER.DEVQ(LINK), N.QUEUE(LINK),
        N.PROP.QUEUE(LINK) AS FOLLOWS
**** ***  ***  *.******    ****.****  *****  ***.****  ****     **
''
    LET MN.LNK = MN.LNK + UTILZ.PERIOD(LINK)
    LET MN.QUE = MN.QUE + PER.QMEAN(LINK)
    LET DEV.QUE = DEV.QUE + PER.DEVQ(LINK)
    IF MX.QU LT PER.MAXQ(LINK),
        LET MX.QU = PER.MAXQ(LINK)
    ALWAYS
LOOP
LET MN.LNK = MN.LNK / N.LINK
LET MN.QUE = MN.QUE / N.LINK
LET DEV.QUE = DEV.QUE / N.LINK
SKIP 1 LINE
PRINT 5 LINES WITH MN.LNK, MN.QUE, MX.QU, DEV.QUE  AS FOLLOWS
----------------------FOR THE NETWORK-----------------------------
              AVERAGE LINK UTILIZATION = **.*****
              AVERAGE QUEUE SIZE = *****.***
              MAXIMUM QUEUE SIZE = ********.
              STD.DEV. OF QUEUE SIZE = ***.****
PRINT 12 LINES WITH MSG.PERIOD, PKT.PERIOD,
 (PKT.PERIOD / MSG.PERIOD), PKTS.ARR.PERIOD, LOOP.PKTS.PERIOD,
 (PKT.TOTAL - COMP.TRIP.PKT - TOTAL.LOOP.PKTS),
 (HOP.PERIOD / PKTS.ARR.PERIOD), (TRIP.PERIOD / PKTS.ARR.PERIOD),
```

```
 (QU.PERIOD / PKTS.ARR.PERIOD), (TRIP.PERIOD / HOP.PERIOD)
 AS FOLLOWS
              NUMBER OF MESSAGES GENERATED = ******.
              NUMBER OF PACKETS GENERATED = *********.
              AVERAGE PACKETS PER MESSAGE = ***.***
              NUMBER OF PACKETS COMPLETED TRIP = *******.
              NUMBER OF PACKETS THAT LOOPED = ******.
              TOTAL NUMBER OF PACKETS STILL IN NETWORK = *******.

------------FOR PACKETS THAT COMPLETED TRIP-------------------
              AVG. LENGTH OF TRIP = ******.*** HOPS
              AVG. TOTAL TRIP TIME = *******.***** SECONDS
              AVG. TIME SPENT IN QUEUE = *****.***** SECONDS
              AVG. TIME PER HOP = *****.***** SECONDS
''
RETURN
END  '' OF PERIOD.REPORT
''
'' ************************************************************
ROUTINE COLLECT.PERIOD.DATA
''
'' THIS ROUTINE CALCULATES AND OUTPUTS SOME PERIOD DATA TO 'UNIT 9'
'' FOR FURTHER GRAPH AND ANALYSIS. STARTING TIME OF PERIOD IS TAKEN
'' FROM 'LAST.PERIOD' AND ENDING TIME FROM 'TIME.V'. ROUTINE
'' 'RESTART.PERIOD.TOTALS' REINITIALIZES TOTALS AND VARIABLES USED
'' IN PERIOD SO DATA OF EACH PERIOD IS INDEPENDENT OF THE REST.
''
'' ------------------------------------------------------------
DEFINE MN.LNK, MN.QUE, MX.QU
    AS REAL VARIABLES
'' ------------------------------------------------------------
''
LET MN.LNK = 0.0
LET MN.QUE = 0.0
LET MX.QU = 0.0
FOR EACH LINK, DO
    LET MN.LNK = MN.LNK + UTILZ.PERIOD(LINK)
    LET MN.QUE = MN.QUE + PER.QMEAN(LINK)
    IF MX.QU LT PER.MAXQ(LINK)
        LET MX.QU = PER.MAXQ(LINK)
    ALWAYS
LOOP
LET MN.LNK = MN.LNK / N.LINK
LET MN.QUE = MN.QUE / N.LINK
''
USE UNIT 9 FOR OUTPUT
   WRITE TIME.V, REAL.F(PKT.TOTAL - COMP.TRIP.PKT - TOTAL.LOOP.PKTS),
      MN.LNK, MN.QUE, REAL.F(MX.QU), (HOP.PERIOD / PKTS.ARR.PERIOD),
```

```
      (TRIP.PERIOD / PKTS.ARR.PERIOD), (QU.PERIOD / PKTS.ARR.PERIOD),
      LOOP.PKTS.PERIOD   AS /,B 1,7 D(9,3),/,2 D(9,3)
USE UNIT 6 FOR OUTPUT
''
PRINT 1 LINE WITH TIME.V AS FOLLOWS
########## COLLECT PERIOD DATA DONE AT TIME = ***.***** ############
''
RETURN
END  '' OF COLLECT.PERIOD.DATA
''
'' **********************************************************
ROUTINE RESTART.PERIOD.TOTALS
''
'' THIS ROUTINE REINITIALIZES TOTALS AND GLOBAL VARIABLES USED
'' IN PERIOD AND SETS PERIOD STARTING TIME.
''
'' ----------------------------------------------------------
''
PRINT 1 LINE WITH TIME.V AS FOLLOWS
### ROUTINE RESTART-PERIOD-TOTALS PERFORMED AT *******.****** SECS ##
LET LAST.PERIOD = TIME.V
LET MSG.PERIOD = 0
LET PKT.PERIOD = 0
LET HOP.PERIOD = 0
LET TRIP.PERIOD = 0
LET QU.PERIOD = 0
LET PKTS.ARR.PERIOD = 0
LET LOOP.PKTS.PERIOD = 0
FOR EACH LINK, DO
    RESET THE PERIODICAL TOTALS OF LNK.STATUS
    RESET THE PERIODICAL TOTALS OF N.QUEUE
LOOP
RETURN
END  '' OF RESTART.PERIOD.TOTALS
''
'' **********************************************************
EVENT NEW.MSG
''
'' THIS EVENT FINDS ORIGIN, DESTINATION AND NUMBER OF PACKETS FOR
'' NEW MESSAGE. IF 'MO.DE = 1' HAS BEEN SELECTED, FINDS BEST LINK,
'' CREATES AND FILES PACKETS IN IT. IF 'MO.DE = 2' HAS BEEN SELECTED
'' FINDS VIRTUAL CIRCUIT, CREATES AND FILES PACKETS IN FIRST LINK OF
'' IT. IF NEW LINK SELECTED IN ANY 'MO.DE' IS IDDLE SCHEDULES AN
'' EVENT 'START TRANSMMISION' FOR IT.
''
'' ----------------------------------------------------------
DEFINE X, I, J, ORIG, DEST, NUM.PKTS, BEST.LNK, STATUS.B.LNK,
    QUEUE.B.LNK, PKT, THE.NAME, REC, LNK, C
```

```
    AS INTEGER VARIABLES
'' ------------------------------------------------------------------
DEFINE Q, H, G, GEOM.SD, R
    AS REAL VARIABLES
'' ------------------------------------------------------------------
''
LET MSG.TOTAL = MSG.TOTAL + 1
LET MSG.PERIOD = MSG.PERIOD + 1
LET X = SELECT.NODES
LET ORIG = TRUNC.F(PR.PAIR(X,2) + 0.1)
LET DEST = TRUNC.F(PR.PAIR(X,3) + 0.1)
''
IF PRNT GE 1,
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE WITH TIME.V AS FOLLOWS
######### EVENT NEW-MESSAGE STARTS AT *******.****** SECS ##########
  PRINT 1 LINE WITH MSG.TOTAL, MSG.PERIOD  AS FOLLOWS
NEW MSG. TOTAL=******          NEW MSG. PERIOD=******
  PRINT 1 LINE WITH X, ORIG, DEST AS FOLLOWS
    X(LOW)=***  MSG.ORIG=***  MSG.DEST=***
ALWAYS
''
IF MO.DE EQ 1,
  LET BEST.LNK = PICK.BEST.LINK(ORIG,DEST)
  IF PRNT GE 1,
     PRINT 3 LINES WITH BEST.LNK, LNK.ORIG(BEST.LNK),
       LNK.TERM(BEST.LNK), QUEUE.B.LNK, N.PROP.QUEUE(BEST.LNK),
       STATUS.B.LNK   AS FOLLOWS
    BEST.LNK=***  ORIG(BEST.LNK)=***       TERM(BEST.LNK)=***
                  N.QUEUE(BEST.LNK)=****  N.PROP.QUEUE(BEST.LNK)=**
                  STATUS(BEST.LNK)=**
  ALWAYS
ALWAYS
IF MC.DE EQ 2,
 RESERVE VIR.CI(*) AS N.LINK
 CALL SET.VIRTUAL.CIRCUIT GIVEN ORIG AND DEST YIELDING C AND VIR.CI(*)
 LET BEST.LNK = VIR.CI(1)
 IF PRNT GE 1,
    PRINT 1 LINE AS FOLLOWS
    VIRTUAL CIRCUIT FOR THIS MESSAGE:
    FOR I=1 TO C, DO
        PRINT 1 LINE WITH I, VIR.CI(I), LNK.ORIG(VIR.CI(I)),
           LNK.TERM(VIR.CI(I))  AS FOLLOWS
                      **) LINK:****   (ORIGIN:*** , TERMINATION:***)
    LOOP
 ALWAYS
ALWAYS
LET STATUS.B.LNK = LNK.STATUS(BEST.LNK)
```

```
LET QUEUE.B.LNK = N.QUEUE(BEST.LNK)
'TRY.AGAIN'
LET GEOM.SD = UNIFORM.F(0.0,1.0,3)
IF (GEOM.SD EQ 0.0) OR (GEOM.SD EQ 1.0),
   GO TRY.AGAIN
REGARDLESS
IF AVG.PKTS.MSG EQ 1.0,
   LET NUM.PKTS = 1
ALWAYS
IF AVG.PKTS.MSG GT 1.0,
   LET G = LOG.E.F(GEOM.SD) / LOG.E.F(1 - (1 / AVG.PKTS.MSG))
   IF G EQ TRUNC.F(G)
      LET NUM.PKTS = G
   ELSE
      LET NUM.PKTS = TRUNC.F(G + 1)
   ALWAYS
   IF NUM.PKTS EQ 0,
      GO TRY.AGAIN
   REGARDLESS
ALWAYS
''
IF PRNT GE 1,
  PRINT 1 LINE WITH NUM.PKTS AS FOLLOWS
    NUM.PKTS FOR THIS MSG=***
ALWAYS
''
FOR I = 1 TO NUM.PKTS, DO
    CREATE A PACKET
        LET PKT = PACKET
        LET PKT.TOTAL = PKT.TOTAL + 1
        LET PKT.PERIOD = PKT.PERIOD + 1
        LET PKT.ID(PACKET) = PKT.TOTAL
        LET PKT.BIRTH(PACKET) = TIME.V
        LET HOP.COUNT(PACKET) = 0
        LET LIFETIME(PACKET) = 1
        LET PKT.ORIG(PACKET) = ORIG
        LET PKT.DEST(PACKET) = DEST
        LET QSTAT(PACKET) = 1
    FILE PACKET IN QUEUE(BEST.LNK)
    LET NET.PKTS = NET.PKTS + 1
    IF MODE EQ 1,
       CREATE A RECORD
          LET PAST.NODE(RECORD) = ORIG
       FILE RECORD IN TRIP.RECORD(PKT)
    ALWAYS
    IF (MODE EQ 2) AND (C GT 1),
       FOR J=2 TO C, DO
           CREATE A RECORD
```

```
            LET PAST.NODE(RECORD) = VIR.CI(J)
          FILE RECORD IN TRIP.RECORD(PKT)
       LOOP
    ALWAYS
''
    IF PRNT GE 1,
       PRINT 8 LINES WITH PKT.ID(PKT), PKT.TOTAL, PKT.BIRTH(PKT),
         PKT.ORIG(PKT), PKT.DEST(PKT), HOP.COUNT(PKT), LIFETIME(PKT),
         QSTAT(PKT)  AS FOLLOWS
NEW-PACKET:
            PKT.ID=******      (PKT.TOTAL=******)
            PKT.BIRTH=******.***
            PKT.ORIG=***
            PKT.DEST=***
            HOP.COUNT=****
            LIFETIME=**
            QSTAT=**
       IF MO.DE EQ 1,
          PRINT 1 LINE AS FOLLOWS
           TRIP RECORD:
          IF TRIP.RECORD(PKT) IS NOT EMPTY,
             FOR EVERY REC IN TRIP.RECORD(PKT), DO
                 PRINT 1 LINE WITH REC, PAST.NODE(REC)  AS FOLLOWS
                       REC #:*******   PAST.NODE=***
             LOOP
          ALWAYS
       ALWAYS
       IF MO.DE EQ 2,
          IF TRIP.RECORD(PKT) IS NOT EMPTY,
             PRINT 1 LINE AS FOLLOWS
           VIRTUAL CIRCUIT NEXT LINKS:
             FOR EVERY REC IN TRIP.RECORD(PKT), DO
                 PRINT 1 LINE WITH  PAST.NODE(REC),
                   LNK.ORIG(PAST.NODE(REC)), LNK.TERM(PAST.NODE(REC))
                   AS FOLLOWS
                          LINK:****   (ORIGIN:*** , TERMINATION:***)
             LOOP
          ELSE
             PRINT 1 LINE AS FOLLOWS
           THIS IS THE ONLY LINK OF ITS TRIP
          ALWAYS
       ALWAYS
    ALWAYS
LOOP
''
IF (STATUS.B.LNK EQ 0) AND (QUEUE.B.LNK EQ 0),
  LET LNK.STATUS(BEST.LNK) = 1
  IF PRNT GE 1,
```

```
      PRINT 1 LINE WITH BEST.LNK, LNK.STATUS(BEST.LNK) AS FOLLOWS
    NEW LNK.STATUS(BL***)=**
  ALWAYS
  SCHEDULE A START.XMT GIVEN BEST.LNK NOW
ALWAYS
IF PRNT GE 2,
  SKIP 1 OUTPUT LINE
  FOR EVERY LINK, DO
      LET LNK = LINK
      PRINT 1 LINE WITH LNK, N.QUEUE(LNK), N.PROP.QUEUE(LNK),
        LNK.STATUS(LNK)  AS FOLLOWS
  LINK***:     N.QUEUE=****       N.PROP.QUEUE=**       LNK.STATUS=**
  LOOP
ALWAYS
LET R = EXPONENTIAL.F((1 / AVG.MPS.NET),8)
IF (TIME.LIMIT - TIME.V) GT R,
  SCHEDULE A NEW.MSG IN R UNITS
  IF PRNT GE 1,
      PRINT 1 LINE WITH R AS FOLLOWS
    EXP-TIME FOR NEXT MESSAGE(R)=**.******
  ALWAYS
ALWAYS
RETURN
END  '' OF NEW.MSG
''
'' **********************************************************************
EVENT START.XMT GIVEN LI.NK
''
'' THIS EVENT REMOVES PACKET FROM QUEUE OF GIVEN LINK AND FILES IT IN
'' PROPAGATION QUEUE. IF LINK TERMINATION EQUALS PACKET DESTINATION
'' SCHEDULES AN EVENT 'ARRIVAL', ELSE SCHEDULES AN EVENT 'END.XMT'
'' FOR THIS LINK.
''
'' ----------------------------------------------------------------
DEFINE LI.NK, PKT
    AS INTEGER VARIABLES
'' ----------------------------------------------------------------
''
REMOVE FIRST PACKET FROM QUEUE(LI.NK)
LET PKT = PACKET
IF PRNT GE 1,
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE WITH TIME.V  AS FOLLOWS
########## EVENT START-XMT STARTS AT *******.****** SECS ##########
  PRINT 1 LINE WITH PKT.ID(PKT), LI.NK AS FOLLOWS
    PACKET(******) REMOVED FROM QUEUE OF LI.NK:***
ALWAYS
LET QSTAT(PKT) = C
```

```
IF PKT.DEST(PKT) = LNK.TERM(LI.NK),
   SCHEDULE AN ARRIVAL GIVEN LI.NK IN
      (LENGTH.PKT / LNK.CAPACITY(LI.NK)) UNITS
   FILE PKT IN PROP.QUEUE(LI.NK)
''
   IF PRNT GE 1,
     PRINT 2 LINES WITH PKT.ID(PKT), LNK.TERM(LI.NK), LI.NK,
       N.QUEUE(LI.NK), N.PROP.QUEUE(LI.NK)
       AS FOLLOWS
         PACKET(******) WILL ARRIVE TO FINAL NODE:***
         LI.NK(***): NEW N.QUEUE=******   NEW N.PROP.QUEUE=**
   ALWAYS
''
   GO OUT
REGARDLESS
SCHEDULE AN END.XMT GIVEN LI.NK IN
    (LENGTH.PKT / LNK.CAPACITY(LI.NK)) UNITS
FILE PKT IN PROP.QUEUE(LI.NK)
''
IF PRNT GE 1,
  PRINT 2 LINES WITH PKT.ID(PKT), LNK.TERM(LI.NK), LI.NK,
       N.QUEUE(LI.NK), N.PROP.QUEUE(LI.NK)
       AS FOLLOWS
         PACKET(********) WILL ARRIVE TO NEXT NODE:****
         LI.NK(****): NEW N.QUEUE=*****   NEW N.PROP.QUEUE=*****
ALWAYS
''
'OUT'
RETURN
END  '' OF START.XMT
''
'' ************************************************************
EVENT END.XMT GIVEN XMT.LINK
''
'' THIS EVENT REMOVES PACKET FROM PROPAGATION QUEUE OF GIVEN LINK
'' IF 'MO.DE = 1' HAS BEEN SELECTED, FINDS NEW BEST LINK AND FILES
'' PACKET IN ITS QUEUE. IF 'MO.DE = 2' HAS BEEN SELECTED FINDS NEXT
'' LINK OF VIRTUAL CIRCUIT AND FILES PACKET IN IT.
'' IF NEW LINK SELECTED IN ANY 'MO.DE' IS IDDLE SCHEDULES AN EVENT
'' 'START TRANSMMISION' FOR IT.
''
'' ------------------------------------------------------------
DEFINE  DEST, BEST.LNK, STATUS.B.LNK, XMT.LINK,
    QUEUE.B.LNK, PKT, REC, LNK
    AS INTEGER VARIABLES
'' ------------------------------------------------------------
''
REMOVE FIRST PACKET FROM PROP.QUEUE(XMT.LINK)
```

```
LET PKT = PACKET
LET DEST = PKT.DEST(PKT)
''
IF PRNT GE 1,
  SKIP 1 OUTPUT LINE
  PRINT 1 LINE WITH TIME.V  AS FOLLOWS
########## EVENT END-XMT STARTS AT *******.****** SECS ############
  PRINT 2 LINES WITH PKT.ID(PKT), XMT.LINK, DEST, LNK.TERM(XMT.LINK)
   AS FOLLOWS
    PACKET(******) REMOVED FROM PROP.QUEUE OF XMT.LINK(***)
                   PKT.DEST=***   LNK.TERM(XMT.LINK)=***
ALWAYS
''
IF MC.DE EQ 1,
  FOR EVERY REC IN TRIP.RECORD(PKT) WITH
      PAST.NODE(REC) = LNK.TERM(XMT.LINK), FIND THE FIRST CASE,
  IF FOUND,
    LET TOTAL.LOOP.PKTS = TOTAL.LOOP.PKTS + 1
    LET LOOP.PKTS.PERIOD = LOOP.PKTS.PERIOD + 1
    IF PRNT GE 1,
      PRINT 4 LINES WITH PKT.ID(PKT), LNK.TERM(XMT.LINK),
        TOTAL.LOOP.PKTS, LOOP.PKTS.PERIOD
        AS FOLLOWS
##########################PACKET LOOPED#############################
    PACKET(******) LOOPED TO NODE:***   NEW TOTAL.LOOP.PKTS=*****
                                        NEW LOOP.PKTS.PERIOD=****
                    ITS TRIP RECORD:
      FOR EVERY REC IN TRIP.RECORD(PKT), DO
        PRINT 1 LINE WITH REC, PAST.NODE(REC) AS FOLLOWS
                                  REC #:*******  PAST.NODE=***
      LOOP
    ALWAYS
    DESTROY PACKET CALLED PKT
    LET NET.PKTS = NET.PKTS - 1
    IF N.QUEUE(XMT.LINK) = 0,
        LET LNK.STATUS(XMT.LINK) = 0
        IF PRNT GE 1,
          PRINT 1 LINE WITH XMT.LINK, LNK.STATUS(XMT.LINK) AS FOLLOWS
    NEW LNK.STATUS OF XMT.LINK(***)=**
        ALWAYS
        GO OUT
    ELSE
        SCHEDULE A START.XMT GIVEN XMT.LINK NOW
        GO OUT
  OTHERWISE
  CREATE A RECORD
     LET PAST.NODE(RECORD) = LNK.TERM(XMT.LINK)
  FILE RECORD IN TRIP.RECORD(PKT)
```

```
   LET BEST.LNK = PICK.BEST.LINK(LNK.TERM(XMT.LINK),DEST)
   IF PRNT GE 1,
     PRINT 2 LINES WITH BEST.LNK, LNK.ORIG(BEST.LNK),
       LNK.TERM(BEST.LNK), STATUS.B.LNK, QUEUE.B.LNK
       AS FOLLOWS
     BEST.LNK=***  ORIG(BEST.LNK)=***  TERM(BEST.LNK)=***
                   STATUS.B.LNK=**     QUEUE.B.LNK=****
   ALWAYS
ALWAYS
IF MC.DE EQ 2,
   REMOVE FIRST RECORD FROM TRIP.RECORD(PKT)
   LET BEST.LNK = PAST.NODE(RECORD)
   DESTROY RECORD
   IF PRNT GE 1,
     PRINT 2 LINES WITH BEST.LNK, LNK.ORIG(BEST.LNK),
       LNK.TERM(BEST.LNK), STATUS.B.LNK, QUEUE.B.LNK
       AS FOLLOWS
     BEST.LNK=***  ORIG(BEST.LNK)=***  TERM(BEST.LNK)=***
                   STATUS.B.LNK=**     QUEUE.B.LNK=****
   ALWAYS
ALWAYS
LET STATUS.B.LNK = LNK.STATUS(BEST.LNK)
LET QUEUE.B.LNK = N.QUEUE(BEST.LNK)
LET HOP.COUNT(PKT) = HOP.COUNT(PKT) + 1
LET QSTAT(PKT) = 1
FILE PKT IN QUEUE(BEST.LNK)
IF PRNT GE 1,
   PRINT 1 LINE WITH N.QUEUE(BEST.LNK) AS FOLLOWS
    NEW QUEUE(BEST.LNK)=****
   IF MO.DE EQ 1,
      PRINT 2 LINES WITH PKT.ID(PKT),HOP.COUNT(PKT),QSTAT(PKT)
         AS FOLLOWS
    PACKET(******):  NEW HOP.COUNT=***** NEW QSTAT=**
                     NEW TRIP RECORD:
      IF TRIP.RECORD(PKT) IS NOT EMPTY,
         FOR EVERY REC IN TRIP.RECORD(PKT), DO
             PRINT 1 LINE WITH REC, PAST.NODE(REC) AS FOLLOWS
                                    REC #:******* , PAST.NODE=***
         LOOP
      ALWAYS
   ALWAYS
   IF MO.DE EQ 2,
      PRINT 1 LINE WITH PKT.ID(PKT),HOP.COUNT(PKT),QSTAT(PKT)
         AS FOLLOWS
    PACKET(******):  NEW HOP.COUNT=***** NEW QSTAT=**
      IF TRIP.RECORD(PKT) IS NOT EMPTY,
         PRINT 1 LINE AS FOLLOWS
                     REST OF VIRTUAL CIRCUIT:
```

```
          FOR EVERY REC IN TRIP.RECORD(PKT), DO
              PRINT 1 LINE WITH  PAST.NODE(REC),
                   LNK.ORIG(PAST.NODE(REC)), LNK.TERM(PAST.NODE(REC))
                   AS FOLLOWS
                              LINK:****   (ORIGIN:*** , TERMINATION:***)
          LOOP
       ELSE
          PRINT 1 LINE AS FOLLOWS
                        THIS IS THE LAST LINK OF ITS TRIP
       ALWAYS
    ALWAYS
ALWAYS
IF (STATUS.B.LNK EQ 0) AND (QUEUE.B.LNK EQ 0),
     LET LNK.STATUS(BEST.LNK) = 1
     IF PRNT GE 1,
       PRINT 1 LINE WITH LNK.STATUS(BEST.LNK) AS FOLLOWS
     NEW LNK.STATUS(BEST.LNK)=**
     ALWAYS
     SCHEDULE A START.XMT GIVEN BEST.LNK NOW
ALWAYS
IF N.QUEUE(XMT.LINK) EQ 0,
     LET LNK.STATUS(XMT.LINK) = 0
     IF PRNT GE 1,
       PRINT 1 LINE WITH XMT.LINK, LNK.STATUS(XMT.LINK) AS FOLLOWS
     NEW LNK.STATUS(XL***)=**
     ALWAYS
ELSE
     SCHEDULE A START.XMT GIVEN XMT.LINK NOW
ALWAYS
''
'OUT'
IF PRNT GE 2,
  FOR EVERY LINK, DO
     LET LNK = LINK
     PRINT 1 LINE WITH LNK, N.QUEUE(LNK), N.PROP.QUEUE(LNK),
       LNK.STATUS(LNK)  AS FOLLOWS
   LINK***:     N.QUEUE=*****      N.PROP.QUEUE=**      LNK.STATUS=**
  LOOP
ALWAYS
''
RETURN
END   '' OF END.XMT
''
'' ************************************************************
EVENT ARRIVAL GIVEN ARR.LNK
''
'' THIS EVENT REMOVES PACKET FROM PROPAGATION QUEUE OF GIVEN LINK
'' RECORDS INFORMATION OF ARRIVING PACKET AND DESTROYS IT.
```

```
''
'' ------------------------------------------------------------------
DEFINE PKT, ARR.LNK
    AS INTEGER VARIABLES
'' ------------------------------------------------------------------
''
REMOVE FIRST PACKET FROM PROP.QUEUE(ARR.LNK)
    LET PKT = PACKET
    LET LIFETIME(PKT) = 0
    LET HOP.COUNT(PKT) = HOP.COUNT(PKT) + 1
''
    IF PRNT GE 1,
      SKIP 1 OUTPUT LINE
      PRINT 1 LINE WITH TIME.V  AS FOLLOWS
######## EVENT PACKET-ARRIVAL STARTS AT *******.****** SECS ########
      PRINT 4 LINES WITH PKT.ID(PKT), ARR.LNK, PKT.DEST(PKT),
        LNK.TERM(ARR.LNK), ARR.LNK,
        N.PROP.QUEUE(ARR.LNK), HOP.COUNT(PKT),
        LIFETIME(PKT)  AS FOLLOWS
PACKET(******)  REMOVED FROM PROP.QUEUE OF ARRIVAL.LNK:***
                PKT.DEST=***   LNK.TERM(ARRIVAL.LNK)=***
                NEW N.PROP.QUEUE(AL***)=**
                NEW HOP.COUNT(PKT)=*****  NEW LIFETIME(PKT)=**
    ALWAYS
''
    LET COMP.TRIP.PKT = COMP.TRIP.PKT + 1
    LET PKTS.ARR.PERIOD = PKTS.ARR.PERIOD + 1
    LET HOP.TOTAL = HOP.TOTAL + HOP.COUNT(PKT)
    LET HOP.PERIOD = HOP.PERIOD + HOP.COUNT(PKT)
    LET TRIP.TOTAL = TRIP.TOTAL + TRANSIT.TIME(PKT)
    LET TRIP.PERIOD = TRIP.PERIOD + TRANSIT.TIME(PKT)
    LET QU.TOTAL = QU.TOTAL + QUE.PKT.TIME(PKT)
    LET QU.PERIOD = QU.PERIOD + QUE.PKT.TIME(PKT)
    LET NET.PKTS = NET.PKTS - 1
''
    IF PRNT GE 1,
     PRINT 4 LINES WITH COMP.TRIP.PKT, HOP.TOTAL, TRIP.TOTAL,
      QU.TOTAL, PKTS.ARR.PERIOD, HOP.PERIOD, TRIP.PERIOD,
      QU.PERIOD   AS FOLLOWS
    NEW COMP.TRIP.PKT=******          NEW HOP.TOTAL=*********
    NEW TRIP.TOTAL=*********.******   NEW QU.TOTAL=*********.*******
    NEW PKTS.ARR.PERIOD=******        NEW HOP.PERIOD=*********
    NEW TRIP.PERIOD=*********.******  NEW QU.PERIOD=*********.*******
    ALWAYS
''
DESTROY PACKET CALLED PKT
IF N.QUEUE(ARR.LNK) = 0,
    IF PRNT GE 1,
```

```
      PRINT 1 LINE WITH ARR.LNK, N.QUEUE(ARR.LNK),
        ARR.LNK, LNK.STATUS(ARR.LNK)  AS FOLLOWS
    N.QUEUE(AL***)=******    STATUS(AL***)=**
    ALWAYS
    LET LNK.STATUS(ARR.LNK) = 0
    IF PRNT GE 1,
     PRINT 1 LINE WITH ARR.LNK,LNK.STATUS(ARR.LNK) AS FOLLOWS
                              NEW STATUS(AL***)=**
    ALWAYS
ELSE
    SCHEDULE A START.XMT GIVEN ARR.LNK NOW
ALWAYS
RETURN
END  '' OF ARRIVAL
''
'' ************************************************************
EVENT NEW.ROUT.FRAC
''
'' THIS EVENT COLLECTS INFORMATION OF QUEUE LENGTH, UTILIZATION PER
'' PERIOD AND UTILIZATION PER LAP OF EVERY LINK.
''
'' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
'' THE PROGRAM TO FIND NEW ROUTING FRACTIONS WHICH IS BEING DEVELOPED
'' AS MATTER OF ANOTHER THESIS SHOULD BE INSERTED OR CALLED FROM THIS
'' EVENT. IT CAN USE THE INFORMATION COLLECTED BY THIS EVENT AND ANY
'' OF THE GLOBAL VARIABLES OF THE PROGRAM LIKE 'LNK.CAPACITY(*)' AND
'' 'CONNECTED(*)', TO FIND THE NEW ROUTING FRACTIONS.
'' @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
''
'' ------------------------------------------------------------
DEFINE I, J
    AS INTEGER VARIABLES
'' ------------------------------------------------------------
''
PERFORM PERIOD.REPORT
PERFORM COLLECT.PERIOD.DATA
IF TIME.LIMIT EQ TIME.V,
   GO OUT
REGARDLESS
RESERVE QSIZE(*) AS N.LINK
RESERVE UTZ.PERIOD(*) AS N.LINK
RESERVE UTZ.LAP(*) AS N.LINK
FOR EVERY LINK, DO
    LET I = LINK
    LET QSIZE(I) = N.QUEUE(LINK)
    LET UTZ.PERIOD(I) = UTILZ.PERIOD(LINK)
    LET UTZ.LAP(I) = LAP.UTILZ(LINK)
LOOP
```

```
PRINT 1 LINE WITH TIME.V  AS FOLLOWS
###### EVENT NEW-ROUTING-FRACTIONS STARTS AT TIME = ***.***** ######
IF PRNT GE 1,
   PRINT 5 LINES AS FOLLOWS

--------------------------- LINKS -------------------------------

   LINK       NODE       NODE       QUEUE       PERIOD          LAP
   NAME       CRIG      TERMIN      SIZE     UTILIZATION   UTILIZATION
   FOR I=1 TO N.LINK, DO
       PRINT 1 LINE WITH I, LNK.ORIG(I), LNK.TERM(I), QSIZE(I),
          UTZ.PERIOD(I), UTZ.LAP(I)  AS FOLLOWS
   ****       ***        ***        ****      **.******      **.******
   LOOP
ALWAYS
''
'' WITH CONNECTED(N.LINK,N.NODE), LNK.CAPACITY(N.LINK),
'' ROUT.FRAC(N.LINK,N.NODE), QSIZE(N.LINK), UTZ.PERIOD(N.LINK),
'' UTZ.LAP(N.LINK)  FIND NEW ROUT.FRAC.
''
IF PRNT GE 1,
   SKIP 1 OUTPUT LINE
   PRINT 4 LINES AS FOLLOWS
--------------------- NEW ROUTING FRACTIONS ------------------------

               FROM        TO        VIA        ROUTING
               NODE       NODE       NODE       FRACTION
   FOR EACH LINK, DO
       FOR J=1 TO N.NODE, DO
           IF ROUT.FRAC(LINK,J) NE 0,
              PRINT 1 LINE WITH LNK.ORIG(LINK), J, LNK.TERM(LINK),
                ROUT.FRAC(LINK,J)
                AS FOLLOWS
               ***        ***        ***         *.****
           ALWAYS
       LOOP
   LOOP
ALWAYS
PERFORM RESTART.PERIOD.TOTALS
IF (TIME.LIMIT-TIME.V) GT UP.TIME,
   SCHEDULE A NEW.ROUT.FRAC IN UP.TIME UNITS
ELSE
   SCHEDULE A NEW.ROUT.FRAC IN (TIME.LIMIT - TIME.V) UNITS
ALWAYS
'OUT'
RETURN
END  '' OF NEW.ROUT.FRAC
''
```

```
'' **************************************************************
EVENT LAP.TOTALS.RESET
''
'' THIS EVENT REINITIALIZES LAP TOTALS OF SIMULATION
''
'' -------------------------------------------------------------
''
PRINT 1 LINE WITH TIME.V AS FOLLOWS
####### EVENT LAP-TOTALS-RESET STARTS AT *******.****** SECS ########
FOR EACH LINK, DO
    RESET THE LAP TOTALS OF LNK.STATUS
LOOP
IF (TIME.LIMIT - TIME.V) GT LAP.TIME,
  SCHEDULE A LAP.TOTALS.RESET IN LAP.TIME UNITS
ALWAYS
RETURN
END   '' OF LAP.TOTALS.RESET
''
'' **************************************************************
EVENT NET.REPORT
''
'' THIS EVENT CALCULATES AND PRINTS STATUS AND STATISTICAL DATA
'' OF SIMULATION UP TO NOW. SINCE VARIABLES AND TOTALS INVOLVED
'' ARE NOT RESET DURING THE RUN, EVERY TIME THIS EVENT IS PERFORMED
'' ITS CALCULATIONS ARE DONE FOR THE PERIOD FROM TIME ZERO UP TO
'' THE TIME OF EXECUTION.
''
'' -------------------------------------------------------------
DEFINE MX.QU, NAME
    AS INTEGER VARIABLES
'' -------------------------------------------------------------
DEFINE MN.LNK, MN.QUE, DEV.QUE, MX.MN.LNK
    AS REAL VARIABLES
'' -------------------------------------------------------------
''
LET MN.LNK = 0.0
LET MN.QUE = 0.0
LET DEV.QUE = 0.0
LET MX.QU = 0
LET MX.MN.LNK = 0.0
SKIP 3 LINES
PRINT 3 LINES WITH TIME.V AS FOLLOWS
############ NETWORK STATUS REPORT AT TIME: ****.***** ############
----------LINK---------  ==========QUEUE SIZE========== ----PROP---
NAME FROM TO  UTILIZATION     MEAN      MAX      DEV      NOW  QUEUE(NOW)
''
FOR EACH LINK, DO
    PRINT 1 LINE WITH LNK.NAME(LINK), LNK.ORIG(LINK),
```

```
        LNK.TERM(LINK), LK.MEAN(LINK),
        QU.MEAN(LINK), QU.MAX(LINK), QU.DEV(LINK),
        N.QUEUE(LINK), N.PROP.QUEUE(LINK) AS FOLLOWS
**** ***  ***  *.******    ****.****  *****  ***.****  ****     **
''
    LET MN.LNK = MN.LNK + LK.MEAN(LINK)
    LET MN.QUE = MN.QUE + QU.MEAN(LINK)
    LET DEV.QUE = DEV.QUE + QU.DEV(LINK)
    IF MX.QU LT QU.MAX(LINK),
        LET MX.QU = QU.MAX(LINK)
    ALWAYS
    IF MX.MN.LNK LT LK.MEAN(LINK),
        LET MX.MN.LNK = LK.MEAN(LINK)
        LET NAME = LINK
    ALWAYS
LOOP
LET MN.LNK = MN.LNK / N.LINK
LET MN.QUE = MN.QUE / N.LINK
LET DEV.QUE = DEV.QUE / N.LINK
SKIP 1 LINE
PRINT 6 LINES WITH MN.LNK, MX.MN.LNK, NAME, MN.QUE, MX.QU, DEV.QUE
   AS FOLLOWS
----------------------FOR THE NETWORK-------------------------------
                AVERAGE LINK UTILIZATION = **.*****
                MAXIMUM LINK UTILIZATION = **.*****(LINK:***)
                AVERAGE QUEUE SIZE = *****.***
                MAXIMUM QUEUE SIZE = *********.
                STD.DEV. OF QUEUE SIZE = ***.****
PRINT 13 LINES WITH MSG.TOTAL, PKT.TOTAL, (PKT.TOTAL / MSG.TOTAL),
    COMP.TRIP.PKT, TOTAL.LOOP.PKTS,
    (PKT.TOTAL - COMP.TRIP.PKT - TOTAL.LOOP.PKTS), AVG.NET.PKTS,
    (HOP.TOTAL / COMP.TRIP.PKT), (TRIP.TOTAL / COMP.TRIP.PKT),
    (QU.TOTAL / COMP.TRIP.PKT), (TRIP.TOTAL / HOP.TOTAL)
    AS FOLLOWS
            NUMBER OF MESSAGES GENERATED = ******.
            NUMBER OF PACKETS GENERATED = *********.
            AVERAGE PACKETS PER MESSAGE = ***.***
            NUMBER OF PACKETS COMPLETED TRIP = *******.
            NUMBER OF PACKETS THAT LOOPED = ******.
            TOTAL NUMBER OF PACKETS STILL IN NETWORK = *******.
            RUNING AVERAGE OF PACKETS IN THE NETWORK = *****.****

-----------FOR PACKETS THAT COMPLETED TRIP--------------------
            AVG. LENGTH OF TRIP = ******.*** HOPS
            AVG. TOTAL TRIP TIME = *******.***** SECONDS
            AVG. TIME SPENT IN QUEUE = *****.***** SECONDS
            AVG. TIME PER HOP = *****.***** SECONDS
''
```

```
IF TIME.LIMIT EQ TIME.V,
   GO OUT
REGARDLESS
IF (TIME.LIMIT - TIME.V) GT REPT.TIME,
    SCHEDULE A NET.REPORT IN REPT.TIME UNITS
ELSE
    SCHEDULE A NET.REPORT IN (TIME.LIMIT - TIME.V) UNITS
ALWAYS
'OUT'
RETURN
END   '' OF NET.REPORT
''
'' ************************************************************
EVENT DATA.COLLECTION
''
'' THIS EVENT CALCULATES AND OUTPUTS SOME SIMULATION DATA TO
'' 'UNIT 8' FOR FURTHER GRAPH AND ANALYSIS. SINCE VARIABLES AND
'' TOTALS INVOLVED ARE NOT RESET DURING THE RUN, EVERY TIME THIS
'' EVENT IS PERFORMED ITS CALCULATIONS ARE DONE FOR THE PERIOD
'' FROM TIME ZERO UP TO THE TIME OF EXECUTION.
''
'' -------------------------------------------------------------
DEFINE MN.LNK, MN.QUE, MX.QU, MX.MN.LNK
    AS REAL VARIABLES
'' -------------------------------------------------------------
''
LET MN.LNK = 0.0
LET MN.QUE = 0.0
LET MX.QU = 0.0
LET MX.MN.LNK = 0.0
FOR EACH LINK, DO
    LET MN.LNK = MN.LNK + LK.MEAN(LINK)
    LET MN.QUE = MN.QUE + QU.MEAN(LINK)
    IF MX.QU LT QL.MAX(LINK)
        LET MX.QU = QU.MAX(LINK)
    ALWAYS
    IF MX.MN.LNK LT LK.MEAN(LINK),
        LET MX.MN.LNK = LK.MEAN(LINK)
        LET NAME = LINK
    ALWAYS
LOOP
LET MN.LNK = MN.LNK / N.LINK
LET MN.QUE = MN.QUE / N.LINK
''
USE UNIT 8 FOR OUTPUT
   WRITE TIME.V, REAL.F(PKT.TOTAL - COMP.TRIP.PKT - TOTAL.LOOP.PKTS),
      MN.LNK, MN.QUE, REAL.F(MX.QU), (HOP.TOTAL / COMP.TRIP.PKT),
      (TRIP.TOTAL / COMP.TRIP.PKT), (QU.TOTAL / COMP.TRIP.PKT),
```

```
      TOTAL.LOOP.PKTS, MX.MN.LNK, AVG.NET.PKTS
      AS /,8 1,7 D(9,3),/,4 D(9,3)
USE UNIT 6 FOR OUTPUT
''
PRINT 1 LINE WITH TIME.V AS FOLLOWS
############ DATA COLLECTION DONE AT TIME = ***.***** ##############
''
IF TIME.LIMIT EQ TIME.V,
   GO OUT
REGARDLESS
IF (TIME.LIMIT - TIME.V) GT COLL.INT,
   SCHEDULE A DATA.COLLECTION IN COLL.INT UNITS
ELSE
   SCHEDULE A DATA.COLLECTION IN (TIME.LIMIT - TIME.V) UNITS
ALWAYS
'OUT'
RETURN
END  '' OF DATA.CCLLECTION
''
'' **************************************************************
EVENT DESTRUCTION
''
'' THIS EVENT CANCELS ALL EVENTS THAT REMAIN SCHEDULED FOR EXECUTION
'' IN THE TIMING ROUTINE, SO AFTER EXECUTION OF THIS EVENT CONTROL
'' OF THE PROGRAM RETURNS TO STATEMENT FOLLOWING 'START SIMULATION'
'' IN THE 'MAIN'.
''
'' ------------------------------------------------------------
''
FOR EACH NEW.MSG IN EV.S(I.NEW.MSG), DO
  CANCEL THE NEW.MSG
  DESTROY THE NEW.MSG
LOOP
FOR EACH START.XMT IN EV.S(I.START.XMT), DO
  CANCEL THE START.XMT
  DESTROY THE START.XMT
LOOP
FOR EACH ARRIVAL IN EV.S(I.ARRIVAL), DO
  CANCEL THE ARRIVAL
  DESTROY THE ARRIVAL
LOOP
FOR EACH END.XMT IN EV.S(I.END.XMT), DO
  CANCEL THE END.XMT
  DESTROY THE END.XMT
LOOP
FOR EACH NET.REPORT IN EV.S(I.NET.REPORT), DO
  CANCEL THE NET.REPORT
  DESTROY THE NET.REPORT
```

```
LOOP
FOR EACH DATA.COLLECTION IN EV.S(I.DATA.COLLECTION), DO
  CANCEL THE DATA.COLLECTION
  DESTROY THE DATA.COLLECTION
LOOP
FOR EACH NEW.ROUT.FRAC IN EV.S(I.NEW.ROUT.FRAC), DO
  CANCEL THE NEW.ROUT.FRAC
  DESTROY THE NEW.ROUT.FRAC
LOOP
FOR EACH LAP.TOTALS.RESET IN EV.S(I.LAP.TOTALS.RESET), DO
  CANCEL THE LAP.TOTALS.RESET
  DESTROY THE LAP.TOTALS.RESET
LOOP
''
SKIP 1 OUTPUT LINE
PRINT 3 LINES WITH TIME.V AS FOLLOWS
---------- DESTRUCTION PERFORMED AT ********.**** SECONDS ----------

########################## END OF SIMULATION ##########################
''
RETURN
END   '' OF DESTRUCTION
''
'' ************************************************************
```

```
C ##################### PLOT NETWORK DATA ##############################
CC
CC  THIS PROGRAM READS THE FILE THAT CONTAINS DATA OUTPUT TO UNIT 8 BY
CC  THE EVENT 'DATA.COLLECTION' OF THE SIMULATION PROGRAM AND PLOTS IT
CC  USING THE DISSPLA GRAPHICS SOFTWARE PACKAGE.
CC
CC  ##################################################################
C
      INTEGER N, NODES, LINKS, LGTPKT, SERVC
      REAL TIME, PKTS, UTILIZ, MQUE, MAXQ, HOP, TRIP, XTIME, QTIME,
     *     LOOP, T, MPS, LIMIT, UPDATE, LAP, DCOLL, PPM, L, MXUTZ
      DIMENSION TIME(60), PKTS(60), UTILIZ(60), MQUE(60), MAXQ(60),
     *     HOP(60), TRIP(60), XTIME(60), QTIME(60), LOOP(60),
     *     MXUTZ(60)
C READ CONDITIONS OF SIMULATION FROM FILE
      READ(5,*) NODES, LINKS, SERVC, LGTPKT, MPS, LIMIT, UPDATE, LAP,
     *     DCOLL, PPM
C PRINT DATA READ
      WRITE(6,100) NODES, LINKS, SERVC, LGTPKT, MPS, LIMIT, UPDATE,
     *     LAP, DCOLL, PPM
C
      N = 34
      T = LIMIT
      L = DCOLL * 3.0
C READ DATA FROM FILE
      DO 10 I=1,N
      READ (5,*) TIME(I), PKTS(I), UTILIZ(I), MQUE(I), MAXQ(I),
     *     HOP(I), TRIP(I), QTIME(I), LOOP(I), MXUTZ(I)
      XTIME(I) = TRIP(I) - QTIME(I)
C PRINT DATA READ
      WRITE (6,200) TIME(I), PKTS(I), UTILIZ(I), MQUE(I), MAXQ(I),
     *     HOP(I), TRIP(I), XTIME(I), QTIME(I), LOOP(I), MXUTZ(I)
   10 CONTINUE
C CALL DISSPLA PLOTTING ROUTINES
      CALL TEK618
      CALL PAGE(17,12)
      CALL NOBRDR
C PLOT PACKETS STILL IN NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('NUMBER OF PACKETS$',100)
      CALL HEADIN('PACKETS STILL IN NETWORK$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,75.0,1200.0)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,PKTS,N,1)
      CALL GRID(1,1)
```

```
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT AVERAGE LINK USAGE FOR NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE LINK USAGE$',100)
      CALL HEADIN('AVERAGE LINK USAGE FOR NETWORK$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,0.1,1.0)
      CALL POLY3
      CALL CURVE(TIME,UTILIZ,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
```

```
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT MAXIMUM-LINK-USAGE FOR NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('MAXIMUM LINK USAGE$',100)
      CALL HEADIN('MAXIMUM-LINK-USAGE$',100,1.,2)
      CALL HEADIN('FOR THE NETWORK$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.1,1.3)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,MXUTZ,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('------------------------>',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
```

```
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT AVERAGE QUEUE LENGTH FOR NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE QUEUE LENGTH(PACKETS)$',100)
      CALL HEADIN('AVERAGE QUEUE LENGTH FOR NETWORK$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,2.0,30.0)
      CALL POLY3
      CALL CURVE(TIME,MQUE,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT MAXIMUM QUEUE LENGTH FOR NETWORK
      CALL PHYSOR(1.0,1.0)
```

```
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('MAXIMUM QUEUE LENGTH(PACKETS)$',100)
      CALL HEADIN('MAXIMUM QUEUE LENGTH FOR NETWORK$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,20.0,300.0)
      CALL POLY3
      CALL CURVE(TIME,MAXQ,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-------------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT HOP AVERAGE FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('HOP AVERAGE(HOPS)$',100)
      CALL HEADIN('HOP AVERAGE FOR COMPLETED-TRIP PACKETS$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,0.3,4.5)
      CALL POLY3
      CALL CURVE(TIME,HOP,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
```

```
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-------------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EC.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EC.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT AVERAGE DELAY FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE DELAY (SECS)$',100)
      CALL HEADIN('AVERAGE DELAY FOR COMPLETED-TRIP PACKETS$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,0.4,6.0)
      CALL POLY3
      CALL CURVE(TIME,TRIP,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-------------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
```

```
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT AVERAGE TRANSMISSION TIME FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE TRANSMISSION TIME (SECS)$',100)
      CALL HEADIN('AVERAGE TRANSMISSION TIME FOR$',100,1.,2)
      CALL HEADIN('COMPLETED-TRIP PACKETS$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.02,0.3)
      CALL POLY3
      CALL CURVE(TIME,XTIME,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABJT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
```

```
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT AVERAGE QUEUE TIME FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE QUEUE TIME (SECS)$',100)
      CALL HEADIN('AVERAGE QUEUE TIME FOR COMPLETED-TRIP$',100,1.,2)
      CALL HEADIN('PACKETS$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.3,4.5)
      CALL POLY3
      CALL CURVE(TIME,XTIME,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
C PLOT NUMBER OF PACKETS THAT LOOPED
      CALL PHYSOR(1.0,1.0)
```

```
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('PACKETS$',100)
      CALL HEADIN('NUMBER OF PACKETS THAT LOOPED$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,10.0,150.0)
      CALL POLY3
      CALL CURVE(TIME,LOOP,N,1)
      CALL MARKER(15)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-------------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL ENDPL(0)
      CALL DONEPL
      STOP
  100 FORMAT(/,2(I4),I2,I5,6(F8.3))
  200 FORMAT(/,9(F8.3),/,2(F8.3))
      END
```

```
C ##################### PLOT PERIOD DATA ############################
C
C THIS PROGRAM READS THE FILE THAT CONTAINS DATA OUTPUT TO UNIT 9 BY
C THE EVENT 'COLLECT.PERIOD.DATA' OF THE SIMULATION PROGRAM AND PLOTS
C IT USING THE DISSPLA GRAPHICS SOFTWARE PACKAGE.
C
C ##################################################################
C
      INTEGER N, NODES, LINKS, LGTPKT, SERVC
      REAL TIME, PKTS, UTILIZ, MQUE, MAXQ, HOP, TRIP, XTIME, QTIME,
     *     LOOP, T, MPS, LIMIT, UPDATE, LAP, L, PPM
      DIMENSION TIME(30), PKTS(30), UTILIZ(30), MQUE(30), MAXQ(30),
     *     HOP(30), TRIP(30), XTIME(30), QTIME(30), LOOP(30)
C READ CONDITIONS OF SIMULATION FROM FILE
      READ(5,*) NODES, LINKS, SERVC, LGTPKT, MPS, LIMIT, UPDATE, LAP,
     *     PPM
C PRINT DATA READ
      WRITE(6,100) NODES, LINKS, SERVC, LGTPKT, MPS, LIMIT, UPDATE,
     *     LAP, PPM
C
      N = 9
      T = LIMIT
      L = UPDATE
C READ DATA FROM FILE
      DO 10 I=1,N
      READ (5,*) TIME(I), PKTS(I), UTILIZ(I), MQUE(I), MAXQ(I),
     *     HOP(I), TRIP(I), QTIME(I), LOOP(I)
      XTIME(I) = TRIP(I) - QTIME(I)
C PRINT DATA READ
      WRITE (6,200) TIME(I), PKTS(I), UTILIZ(I), MQUE(I), MAXQ(I),
     *     HOP(I), TRIP(I), XTIME(I), QTIME(I), LOOP(I)
   10 CONTINUE
C CALL DISSPLA PLOTTING ROUTINES
      CALL TEK618
      CALL PAGE(17,12)
      CALL NOBRDR
C PLOT TOTAL NUMBER OF PACKETS STILL IN NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('NUMBER OF PACKETS$',100)
      CALL HEADIN('TOTAL NUMBER OF PACKETS STILL IN NETWORK$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,75.0,1200.0)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,PKTS,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
```

```
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-------------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT PERIODICAL AVERAGE-LINK-USAGE FOR NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE LINK USAGE$',100)
      CALL HEADIN('PERIODICAL AVERAGE-LINK-USAGE$',100,1.,2)
      CALL HEADIN('FOR THE NETWORK$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.1,1.0)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,UTILIZ,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-------------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
```

```
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT PERIODICAL AVERAGE-QUEUE-LENGTH FOR NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE QUEUE LENGTH(PACKETS)$',100)
      CALL HEADIN('PERIODICAL AVERAGE-QUEUE-LENGTH$',100,1.,2)
      CALL HEADIN('FOR THE NETWORK$',100,1.,2)
      CALL GRAF(0.0,1.,T,0.0,2.0,30.0)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,MQUE,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
```

```
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT PERIODICAL MAXIMUM-QUEUE-LENGTH FOR NETWORK
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('MAXIMUM QUEUE LENGTH(PACKETS)$',100)
      CALL HEADIN('PERIODICAL MAXIMUM-QUEUE-LENGTH$',100,1.,2)
      CALL HEADIN('FOR THE NETWORK$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,20.0,300.0)
      CALL PCLY3
      CALL MARKER(15)
      CALL CURVE(TIME,MAXQ,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
```

```
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT PERIODICAL HOP-AVERAGE FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('HOP AVERAGE(HOPS)$',100)
      CALL HEADIN('PERIODICAL HOP-AVERAGE FOR COMPLETED-TRIP$',100,1.,2)
      CALL HEADIN('PACKETS$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.3,4.5)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,HOP,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('----------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT PERIODICAL AVERAGE-DELAY FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE DELAY (SECS)$',100)
```

```
      CALL HEADIN('PERIODICAL AVERAGE-DELAY FOR$',100,1.,2)
      CALL HEADIN('COMPLETED-TRIP-PACKETS$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.4,6.0)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,TRIP,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT PERIODICAL AVERAGE-TRANSMISSION-TIME FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE TRANSMISSION TIME (SECS)$',100)
      CALL HEADIN('PERIODICAL AVERAGE-TRANSMISSION-TIME FOR$',100,1.,2)
      CALL HEADIN('COMPLETED-TRIP PACKETS$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.02,0.3)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,XTIME,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
```

```
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT PERIODICAL AVERAGE-QUEUE-TIME FOR COMPLETED-TRIP PACKETS
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('AVERAGE QUEUE TIME (SECS)$',100)
      CALL HEADIN('PERIODICAL AVERAGE-QUEUE-TIME FOR$',100,1.,2)
      CALL HEADIN('COMPLETED-TRIP PACKETS$',100,1.,2)
      CALL GRAF(0.0,L,T,0.0,0.1,1.5)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,XTIME,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-----------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
```

```
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(0)
C PLOT NUMBER OF PACKETS THAT LOOPED BY PERIOD
      CALL PHYSOR(1.0,1.0)
      CALL AREA2D(12.0,9.5)
      CALL XNAME('TIME (SECS)$',100)
      CALL YNAME('PACKETS$',100)
      CALL HEADIN('NUMBER OF PACKETS THAT LOOPED BY PERIOD$',100,1.,1)
      CALL GRAF(0.0,L,T,0.0,10.0,150.0)
      CALL POLY3
      CALL MARKER(15)
      CALL CURVE(TIME,LOOP,N,1)
      CALL GRID(1,1)
C ADD CONDITIONS OF SIMULATION
      CALL MESSAG('CONDITIONS OF SIMULATION$',100,12.8,9.0)
      CALL MESSAG('-------------------------$',100,12.8,8.6)
      CALL MESSAG('NUMBER OF NODES : $',100,12.5,8.0)
      CALL INTNO(NODES,'ABUT','ABUT')
      CALL MESSAG('NUMBER OF LINKS : $',100,12.5,7.5)
      CALL INTNO(LINKS,'ABUT','ABUT')
      CALL MESSAG('AVG MESSAGES PER SEC : $',100,12.5,7.0)
      CALL REALNO(MPS,2,'ABUT','ABUT')
      CALL MESSAG('AVG PACKETS PER MSG. : $',100,12.5,6.5)
      CALL REALNO(PPM,2,'ABUT','ABUT')
      CALL MESSAG('PACKET LENGTH : $',100,12.5,6.0)
      CALL INTNO(LGTPKT,'ABUT','ABUT')
      CALL MESSAG(' BITS$',100,'ABUT','ABUT')
      CALL MESSAG('SERVICE : $',100,12.5,5.5)
      IF (SERVC.EQ.1) CALL MESSAG('DATAGRAM$',100,'ABUT','ABUT')
      IF (SERVC.EQ.2) CALL MESSAG('VIRTUAL CIRCUIT$',100,'ABUT','ABUT')
      CALL MESSAG('DURATION OF RUN : $',100,12.5,5.0)
```

```
      CALL REALNO(LIMIT,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('UPDATE INTERVAL : $',100,12.5,4.5)
      CALL REALNO(UPDATE,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('LAP INTERVAL : $',100,12.5,4.0)
      CALL REALNO(LAP,2,'ABUT','ABUT')
      CALL MESSAG('SECS$',100,'ABUT','ABUT')
      CALL MESSAG('(PERIOD-LENGTH = UPDATE-INT.)$',100,12.5,3.5)
      CALL ENDPL(C)
      CALL DONEPL
      STOP
  100 FORMAT(/,2(I4),I2,I5,5(F8.3))
  200 FORMAT(/,9(F8.3),/,F8.3)
      END
```

LIST OF REFERENCES

1. Kermani P. and Kleinrock L., "A Tradeoff Study of Switching System in Computer Communication Networks", IEEE Trans. on Computers, Vol. C-29, No. 12, Dec., 1980.

2. Gold B., "Digital Speech Networks", Proc. IEEE, Vol. 65, Dec., 1977.

3. Ulug M. E. and Gruber J. G., "Statistical Multiplexing of Data and Encoded Voice in a Transparent Intelligence Network", Proc. Fifth Data Comm. Symposium, Sept., 1977.

4. Nemeth A.G., "Behavior of a Link in a PVC Network", Lincoln Laboratory ESD-TR-76-338, Dec., 1976.

5. Tanenbaum A., Computer networks, Prentice-Hall Inc., New Jersey, 1981.

6. Gerla M., "The Design of Store-and-Forward Networks for Computer Communications", Ph.D. Dissertation, School of Engineering and Applied Science, University of California, Los Angeles, Jan., 1973.

7. Schwartz M. and Stern T., "Routing Techniques Used in Computer Communication Networks", IEEE Trans. on Comm., Vol. Com-28, No. 4, April, 1980.

8. Chou W., Eckl J., Frank H. and Gerla M., "A Cut-Saturation Algorithm for Topological Design of Packet-Switched Communications", Proc. IEEE National Telecom. Conference, San Diego, Dec., 1973.

9. Rudin H. "On Routing and Delta Routing: A Taxonomy and Performance Comparison of Techniques for Packet-Switched Networks", IEEE Trans. on Comm., Vol. COM-24, No. 1, Jan., 1976.

10. Fratta L., Gerla M. and Kleinrock L., "The Flow Deviation Method", Networks, Vol. 3, No. 3, 1973.

11. Yum T.S., "Adaptive Routing in Computer Communication Networks", Ph.D. Dissertation, Columbia University, 1978.

12. Kerr I.A. et al, "A Simulation Study of Routing and Flow Control Problems In a Hierarchically Connected Packet Switching Network", Proc. ICCC, 1976.

13. McQuillan J. M., "Adaptive Routing Algorithms for Distributed Computer Networks", Bolt, Beranek and Newman, Report 2831, May, 1974.

14. Chu W.W. and Shen M.Y."A Hierarchical Routing and Flow Control Policy for Packet-Switched Networks", Proc. Int. Symposium on Computer Performance Modeling, Measurement and Evaluation, Yorktown Heights, Aug., 1977.

15. Rudin H., "Chairman's Remarks: An Introduction to Flow Control.", Proc. Int. Symp. Flow Control in Computer Networks, Versailles, France, 1979.

16. Pennotti M.C. and Schwartz M., "Congestion Control in Store and Forward Tandem Links", IEEE Trans. on Comm., Vol. COM-23, No. 12, Dec., 1975.

17. Saad S. and Schwartz M., "Analysis of Congestion Control Techniques in Computer Communication Networks", Proc. Int. Symp. Flow Control in Computer Networks, Versailles, France, 1979.

18. Irland M. I., "Analysis and Simulation of Congestion in Packet Switched Networks", University of Waterloo, Comp. Comm. Network Group, Rpt. E-61, April, 1977.

19. Davies D. W., "The Control of Congestion in Packet Switching Networks", IEEE Trans. on Comm., Vol. COM-20, No. 3, June, 1972.

20. Kleinrock L., Communication Nets: Stochastic Message Flow and Delay, New York, Mc Graw-Hill, 1964.

21. Kleinrock L., "Analytic and Simulation Methods in Computer Network Design", in Conf. Rec. Spring Joint Comput. Conf., AFIPS Conf. Proc., pp. 569-579, 1970.

22. Segall A., "The Modelling of Adaptive Routing in Data Communication Networks", IEEE Trans. on Comm., Vol. COM-25, No. 1, Jan., 1977.

23. Moss F., "The Application of Optimal Control Theory to Dynamic Routing in Data Communication Networks", Ph.D. Dissertation, Massachusetts Inst.Tech., Cambridge, 1976.

24. Wozencraft J.M., Gallager R.G., and Segal A., "First Annual Report on Data Network Reliability", ESL Internal Report ESL-IR-677, M.I.T., July, 1976.

25. Ros F., "Routing to Minimize the Maximum Congestion in a Communication Network", Ph.D. Dissertation, M.I.T., 1978.

26. Older W., Qualitative Analysis of Congestion Sensitive Routing, IFIP North-Holland Publishing Company, 1979.

27. McQuillan J., Richer I. and Rosen E., "The New Routing Algorithm for the Arpanet", IEEE Trans. on Comm., Vol. COM-28, No. 5, May, 1980.

28. Gallager R., "A Minimum Delay Routing Algorithm, Using Distributed Computation", IEEE Trans. on Comm., Vol. Com-25, No. 1, Jan., 1977.

29. Little D.C., "A Proof of the Queueing Formula: L=lW", Operations Research, Vol 9, 1961.

BIBLIOGRAPHY

Chretien G. J., Leyden N. V., NATO Advanced Study Institute Series, pp. 373-376, Netherlands, 1975.

McQuillan M., "Interactions Between Routing and Congestion Control in Computer Networks", Proc. Int. Symp. Flow Control in Computer Networks, Versailles, France, Feb., 1979.

Frank H. and Chow W., "Routing in Computer Networks", Networks, Vol.1, pp. 99-122, 1971.

Slyke R., Chow W. and Frank H., "Avoiding Simulation in Simulating Computer Communication Networks", Proc. National Computer Conf. 165, 1973.

Feit A., "Dynamic Multicommodity Flow Schedules", Ph.D. Dissertation, Naval Postgraduate School, Monterey, 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
3.	Department Chairman, Code 62 Department of Electrical Engineering Naval Postgraduate School Monterey, California 93943	1
4.	Prof. J. M. Wozencraft, Code 62Wn Department of Electrical Engineering Naval Postgraduate School Monterey, California 93943	3
5.	Lieutenant Commander Harry Thornberry Peruvian Navy Clemente X 335 San Isidro Lima, Peru	2
6.	Lieutenant Colonel Kuon Sik Bak Korean Army SMC Box 1328 Naval Postgraduate School Monterey, California 93943	1
7.	Lieutenant Petros Magoulas Helenic Air Force 41 Kilkis Street Patras, Greece	1

Thesis
S33644
c.1

Thesis
S33644
c.1

207744

Schiantarelli
Multiple path static routing protocols for packet switched networks.

30588

02 JAN 1986 SWAS
1 MAY 89 33632
1 MAY 89 33632

207744

Thesis
S33644
c.1

Schiantarelli
Multiple path static routing protocols for packet switched networks.

thesS33644
Multiple path static routing protocols f
3 2768 002 00353 5
DUDLEY KNOX LIBRARY